

UNIVERSIDADE TECNOLÓGICA FEDERAL DO PARANÁ

NICOLAS CARDOSO MOTTA

VINICIUS APARECIDO DE OLIVEIRA HASHIZUME

**RESOLUÇÃO DE PROBLEMA POR MEIO DE ALGORITMOS DE BUSCA:
JOGO PAC-MAN**

PONTA GROSSA

2025

NICOLAS CARDOSO MOTTA
VINICIUS APARECIDO DE OLIVEIRA HASHIZUME

RESOLUÇÃO DE PROBLEMA POR MEIO DE ALGORITMOS DE BUSCA:
JOGO PAC-MAN

Trabalho para a matéria de Inteligência Artificial
apresentada como requisito para a conclusão da
matéria da Universidade Tecnológica Federal do
Paraná (UTFPR).
Orientador(a): Helyane Bronoski Borges

PONTA GROSSA
2025

RESUMO

Este trabalho apresenta o desenvolvimento de uma adaptação do jogo Pac-Man com aplicação de técnicas de Inteligência Artificial para controle dos agentes adversários (fantasmas). Utilizando a plataforma GameMaker, implementou-se algoritmos de busca heurística (A* e busca gulosa) com heurísticas específicas para cada fantasma, reproduzindo seus comportamentos característicos do jogo original. O projeto incluiu três níveis com labirintos distintos, onde os agentes adaptam suas estratégias dinamicamente. Os resultados demonstram a eficácia da abordagem, com os fantasmas exibindo comportamentos coerentes, validando o uso de técnicas de busca em espaços de estados para jogos. O trabalho serviu como ponte entre teoria e prática, consolidando conceitos de IA enquanto mantém a jogabilidade.

Palavras-chave: Pac-Man, Artificial Intelligence, Search Algorithms, A*, Greedy Search, GameMaker.

ABSTRACT

This work presents the development of a Pac-Man adaptation applying Artificial Intelligence techniques to control adversarial agents (ghosts). Using the GameMaker platform, heuristic search algorithms (A* and Greedy Best-First) with specific heuristics for each ghost were implemented, reproducing their characteristic behaviors from the original game. The project included three levels with distinct mazes, where agents dynamically adapt their strategies. Results demonstrate the approach's effectiveness, with ghosts exhibiting coherent, validating state-space search techniques for games. The project bridged theory and practice, consolidating AI concepts while preserving the gameplay.

Keywords: Pac-Man, Artificial Intelligence, Search Algorithms, A*, Greedy Search, GameMaker.

LISTA DE ILUSTRAÇÕES

Captura de Tela 1 - Evento criar do objeto player	13
Captura de Tela 2 - Evento etapa do objeto player (1)	14
Captura de Tela 3 - Evento etapa do objeto player (2)	14
Captura de Tela 4 - Evento alarme 0 do objeto player	15
Captura de Tela 5 - Evento alarme 1 do objeto player	16
Captura de Tela 6 - Evento colisão do objeto player com o objeto Pinky	17
Captura de Tela 7 - Evento colisão do objeto player com o objeto ponto	18
Captura de Tela 8 - E. colisão do objeto player com o objeto Power Pellet	19
Captura de Tela 9 - Evento término da animação do objeto player	20
Captura de Tela 10 - Script da função heuristic	21
Captura de Tela 11 - Script da função astarpath (1)	22
Captura de Tela 12 - Script da função astarpath (2)	23
Captura de Tela 13 - Script da função busca gulosa (1)	24
Captura de Tela 14 - Script da função buscagulosa (2)	24
Captura de Tela 15 - E. de evento de criação dos objetos dos fantasmas	25
Captura de Tela 16 - Evento etapa do objeto blinky (1)	27
Captura de Tela 17 - Evento etapa do objeto blinky (2)	27
Captura de Tela 18 - Evento etapa do objeto pinky	29
Captura de Tela 19 - Evento etapa do objeto clyde	31
Captura de Tela 20 - Evento etapa do objeto clyde (2)	32
Captura de Tela 21 - Evento criação do objeto controlador	34
Captura de Tela 22 - Evento etapa do objeto controlador	35
Captura de Tela 23 - Evento desenhar do objeto controlador	36
Captura de Tela 24 - Tela Inicial	37
Captura de Tela 25 - Tela de Opções	38
Captura de Tela 26 - Tela de Instruções Iniciais	38
Captura de Tela 27 - Primeira Fase	40
Captura de Tela 28 - Segunda Fase	41
Captura de Tela 29 - Terceira Fase	42
Captura de Tela 30 - Tela de Gameover	43
Captura de Tela 31 - Tela Final	44

SUMÁRIO

1	INTRODUÇÃO.....	7
1.1	Descrição do Pac-Man.....	7
1.2	Descrição do Problema.....	8
2	DESENVOLVIMENTO.....	9
2.1	Metodologia.....	9
2.1.1	<u>Algoritmos dos Fantasmas.....</u>	10
2.2	Implementação.....	11
2.2.1	Objeto Player.....	13
2.2.2	Scripts.....	20
2.2.2.1	<u>Função Heurística.....</u>	20
2.2.2.2	<u>Função A estrela.....</u>	21
2.2.2.3	<u>Função Busca Gulosa.....</u>	23
2.2.3	Fantasmas.....	25
2.2.4	Controlador.....	33
2.3	Resultados.....	44
3	CONCLUSÃO.....	45
	REFERÊNCIAS.....	46

1 INTRODUÇÃO

Este trabalho tem como objetivo o desenvolvimento de uma versão do jogo *Pac-Man* com tema de cidades inteligentes, com ênfase na aplicação de técnicas de Inteligência Artificial para controlar o comportamento dos fantasmas, agentes adversários que interagem com o jogador. O projeto propõe a modelagem do ambiente do jogo como um espaço de estados e utiliza algoritmos de busca heurística para simular a tomada de decisão autônoma desses agentes.

A implementação foi realizada na plataforma *GameMaker*, que ofereceu os recursos necessários para a criação da lógica do jogo, controle dos agentes e manipulação da interface gráfica adaptada ao contexto. Durante o processo de desenvolvimento, foram utilizados conteúdos de terceiros e ferramentas baseadas em Inteligência Artificial como apoio para a resolução de desafios técnicos e conceituais. Essa abordagem permitiu uma compreensão mais aprofundada dos algoritmos envolvidos e viabilizou a execução do projeto.

Cada um dos quatro fantasmas foi associado a uma combinação distinta de algoritmo e heurística, respeitando suas características individuais. Os algoritmos de busca A* e busca gulosa foram aplicados para guiar as decisões autônomas dos fantasmas, proporcionando comportamentos variados e estratégicos. A movimentação do Pac-Man, por sua vez, é controlada diretamente pelo jogador.

O jogo foi estruturado com três níveis, cada um com um labirinto distinto, o que exigiu adaptações nas estratégias dos fantasmas em tempo real. Os resultados obtidos demonstram que a aplicação de técnicas de busca heurística resultou em agentes adversários mais desafiadores e coerentes com os padrões do jogo original. Além disso, o projeto permitiu consolidar, na prática, os conhecimentos adquiridos sobre representação de problemas, definição de heurísticas e implementação de algoritmos de busca em ambientes interativos com a temática proposta.

1.1 Descrição do Pac-Man

Pac-Man é um clássico dos jogos eletrônicos, lançado originalmente em 1980 pela empresa japonesa Namco e desenvolvido pelo designer Toru Iwatani. Seu objetivo era criar um jogo que atraísse tanto o público masculino quanto o feminino, em contraste com os populares jogos de tiro da época, voltados majoritariamente para homens. A inspiração para o personagem surgiu de forma inusitada: durante um jantar, ao observar uma pizza com uma fatia faltando, Iwatani visualizou uma boca aberta, o que levou à criação do icônico personagem. Inicialmente nomeado

Puck-Man — referência ao termo japonês "paku-paku", que remete ao ato de mastigar — o jogo teve seu nome alterado para *Pac-Man* no lançamento internacional, a fim de evitar trocadilhos ofensivos em inglês.

No jogo, o jogador controla *Pac-Man*, um personagem representado por um círculo amarelo com uma boca que se abre e fecha, cuja missão é comer todos os *pac-dots* (pontos amarelos) espalhados por um labirinto. Durante esse percurso, o jogador deve evitar ser capturado por quatro fantasmas inimigos. O labirinto é composto por corredores e passagens estreitas, incluindo quatro “*power pellets*” — pílulas de poder — localizadas nos cantos. Ao consumir uma *power pellet*, os fantasmas tornam-se temporariamente vulneráveis, ficando azuis e podendo ser devorados por *Pac-Man*, o que concede pontos extras ao jogador. Após alguns segundos, os fantasmas retornam ao estado normal e retomam a perseguição.

Cada um dos quatro fantasmas possui padrões de movimento e comportamentos distintos, o que adiciona uma camada estratégica ao jogo:

- Blinky (vermelho): persegue diretamente *Pac-Man* e acelera à medida que mais *pac-dots* são consumidos.
- Pinky (rosa): tenta antecipar os movimentos de *Pac-Man*, movendo-se para onde ele está prestes a ir.
- Inky (azul claro): exibe um comportamento imprevisível, combinando elementos das estratégias de Blinky e Pinky.
- Clyde (laranja): alterna entre perseguição e fuga, recuando para um canto do labirinto quando se aproxima demais de *Pac-Man*.

O jogador acumula pontos ao comer *pac-dots*, frutas que surgem ocasionalmente no centro do labirinto e fantasmas vulneráveis. O jogo progride por fases, cada uma com layout distinto e dificuldade crescente — à medida que o jogador avança, os fantasmas se tornam mais rápidos e inteligentes. O objetivo principal é maximizar a pontuação, sobrevivendo o maior tempo possível diante da crescente complexidade do desafio.

1.2 Descrição do Problema

A técnica de busca em espaços de estados é uma abordagem fundamental em Inteligência Artificial, na qual agentes são capazes de executar ações que modificam seu estado atual a fim de alcançar um objetivo. Neste projeto, propomos a implementação de uma versão do jogo *Pac-Man*, modelando o agente principal (*Pac-Man*) e quatro agentes adversários (os fantasmas) dentro desse paradigma. O

agente *Pac-Man* será controlado diretamente pelo jogador, que poderá movimentá-lo em quatro direções (cima, baixo, esquerda e direita). Já os fantasmas serão controlados por algoritmos de busca heurística, representando agentes inteligentes com comportamentos distintos.

A proposta inclui a criação de, no mínimo, três níveis de jogo, cada um com um labirinto diferente, exigindo adaptações nas estratégias de busca. Para cada um dos quatro fantasmas, será definida uma heurística específica, levando em consideração suas características individuais. Pretende-se aplicar os algoritmos de busca A^* e busca gulosa (*Greedy Best-First Search*), sendo exigido que ambos sejam utilizados ao longo do jogo. A escolha do algoritmo e da heurística para cada agente ficará a critério do desenvolvedor, desde que os dois algoritmos estejam representados na implementação final.

2 DESENVOLVIMENTO

Nesta seção, serão descritos os processos envolvidos na construção do projeto, desde as decisões iniciais de implementação até a aplicação dos algoritmos de Inteligência Artificial nos agentes do jogo. O desenvolvimento foi guiado por etapas, incluindo a criação da estrutura base do jogo na ferramenta *Game Maker Studio 2*, a modelagem do comportamento dos fantasmas, e a definição das heurísticas e algoritmos de busca utilizados. A seguir, detalha-se a metodologia adotada para a resolução do problema proposto.

2.1 Metodologia

A metodologia adotada neste projeto foi construída de forma iterativa, acompanhando as mudanças de ferramentas conforme os desafios técnicos encontrados ao longo do processo. Inicialmente, buscou-se utilizar a plataforma Unity para o desenvolvimento do jogo, devido à sua robustez e suporte avançado a jogos 2D e 3D. No entanto, a complexidade da ferramenta e sua elevada curva de aprendizado inviabilizaram sua adoção dentro do tempo disponível para o projeto.

A segunda tentativa foi com o uso da linguagem Python em conjunto com a biblioteca Pygame. Embora mais acessível, a biblioteca apresentou diversas limitações técnicas, especialmente na detecção de colisões e no controle refinado dos movimentos dos fantasmas, que são elementos fundamentais para o funcionamento do jogo. Devido a esses obstáculos, decidiu-se migrar para a plataforma GameMaker, que se mostrou uma solução intermediária entre a simplicidade do Pygame e a robustez da Unity.

O *GameMaker*, foi possível avançar na construção do projeto de forma mais eficiente. A primeira etapa consistiu na implementação da lógica básica do jogo, com foco na movimentação do *Player* e na geração aleatória dos inimigos na tela, com comportamentos iniciais puramente randômicos. A partir dessa base funcional, foi desenvolvido o primeiro labirinto do jogo, utilizado para testes iniciais de movimentação e colisão.

Com o ambiente de teste funcional, o foco se voltou para o desenvolvimento da lógica de comportamento dos fantasmas, que constitui o principal objetivo deste trabalho. Cada um dos quatro fantasmas foi associado a um algoritmo de busca e a uma heurística específica, a fim de representar suas personalidades distintas e simular tomadas de decisão autônomas dentro do jogo.

A seguir, detalha-se a aplicação dos algoritmos utilizados em cada agente.

2.1.1 Algoritmos dos Fantasmas

Para implementar os comportamentos inteligentes dos fantasmas no jogo, foram utilizados dois algoritmos de busca amplamente conhecidos: A estrela (A^*) e busca gulosa. Ambos têm como objetivo encontrar o melhor caminho até um destino em um ambiente com obstáculos, como o labirinto do jogo, mas adotam estratégias diferentes para essa finalidade.

O algoritmo A^* (ou *A-Star*) é uma técnica de busca heurística que combina dois critérios para tomar decisões: o custo real do caminho já percorrido até o ponto atual ($g(n)$) e uma estimativa da distância até o destino ($h(n)$). A soma dessas duas informações define a função $f(n) = g(n) + h(n)$, que guia a escolha do próximo nó a ser explorado. Essa abordagem permite que o algoritmo encontre o caminho mais curto de forma eficiente, mesmo em ambientes complexos.

Já a busca gulosa é uma técnica mais simples e rápida, que se baseia apenas na estimativa de distância até o objetivo ($h(n)$), ignorando o custo do caminho já percorrido. Ou seja, a cada etapa, o algoritmo escolhe o próximo movimento com base na posição que parece estar mais próxima do destino, sem garantir que o trajeto será o mais curto.

Blinky, o fantasma vermelho, é caracterizado por perseguir diretamente o Pac-Man durante todo o jogo, adotando um comportamento de perseguição constante. Para reproduzir esse comportamento de forma eficiente, foi implementado o algoritmo de busca A^* , que calcula o caminho ideal até a posição atual do Pac-Man. A cada atualização, o algoritmo avalia os caminhos possíveis no

labirinto considerando tanto o custo já percorrido quanto uma estimativa da distância até o jogador, evitando rotas com obstáculos e priorizando o trajeto mais curto.

Pinky, o fantasma rosa, apresenta um comportamento mais estratégico: em vez de seguir diretamente o Pac-Man, ela tenta se posicionar à frente dele, antecipando seus movimentos. Para isso, foi utilizado o algoritmo de busca gulosa, que se baseia exclusivamente em uma estimativa heurística da distância até um alvo. No código, essa posição-alvo é calculada projetando a localização do Pac-Man alguns blocos à frente na direção em que ele está se movendo. A busca gulosa, então, determina a melhor direção a seguir em direção a esse ponto estimado, sem considerar o caminho já percorrido.

Clyde, o fantasma laranja, apresenta um comportamento híbrido, alternando entre perseguir o Pac-Man e se afastar dele, dependendo da distância entre ambos. No jogo, quando Clyde está longe do Pac-Man, ele se comporta como Blinky, utilizando o algoritmo A* para persegui-lo diretamente. No entanto, ao se aproximar demais (com uma distância de 4 blocos) ele muda sua estratégia e passa a se mover em direção a uma área fixa do mapa, fugindo do jogador.

Inky, o fantasma azul, apresenta um comportamento imprevisível, resultante da combinação entre as estratégias de Blinky e Pinky. No jogo, sua lógica alterna entre perseguir diretamente o Pac-Man, como faz Blinky (a partir das coordenadas do mesmo) e tentar antecipar os movimentos do jogador, como Pinky, por meio da busca gulosa.

2.2 Implementação

A implementação do projeto no *GameMaker Studio 2* foi estruturada com base na divisão do jogo em múltiplos objetos, cada um com responsabilidades específicas. Os principais elementos desenvolvidos incluem: o jogador (Pac-Man), os quatro fantasmas com comportamentos distintos (Blinky, Pinky, Inky e Clyde), um objeto controlador responsável pelo gerenciamento de funções globais e pelo armazenamento de variáveis compartilhadas, além de objetos dedicados à representação dos pontos normais e pontos de poder, das paredes do labirinto e das interfaces de menu e tela de game over. Complementarmente, foram desenvolvidos scripts específicos para a implementação dos algoritmos de busca heurística, incluindo a Busca Gulosa e o Algoritmo A* (A estrela). A abordagem adotada para cada um desses algoritmos será detalhada nas subseções seguintes.

No contexto da linguagem *GameMaker Language (GML)*, os objetos são estruturados por meio de eventos, que representam momentos específicos no ciclo de vida e na interação dos objetos durante a execução do jogo. Cada evento permite que comportamentos distintos sejam definidos, promovendo organização e modularidade na lógica de programação. Na implementação deste projeto, foram utilizados os seguintes eventos principais:

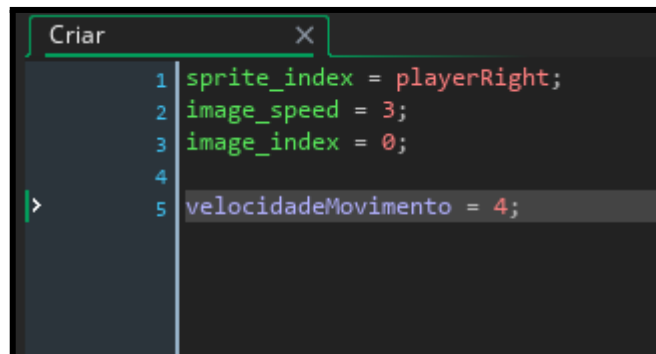
- *Create* (Criar): Executado uma única vez, no momento em que o objeto é instanciado no jogo. Neste evento, são inicializadas variáveis, definidos estados iniciais (como posição e velocidade) e carregadas referências importantes para o comportamento dos personagens e elementos do jogo.
- *Destroy* (Destruir): Acionado quando o objeto é removido da instância do jogo. Foi utilizado para liberar recursos, redefinir variáveis globais ou atualizar contadores que dependem da existência de determinados objetos (por exemplo, pontos coletáveis).
- *Step* (Etapa): Executado continuamente a cada frame do jogo, sendo responsável pela atualização da lógica em tempo real. Nesse evento, foram implementadas funções de movimentação, verificação de condições de jogo.
- *Alarm* (Alarme): Trata contadores regressivos que disparam ações após um determinado número de frames. No projeto, foi usado para controlar o tempo de efeito dos pontos de poder (como o tempo que os fantasmas ficam vulneráveis) ou para realizar ações temporizadas, como retomar a movimentação após uma pausa.
- *Collision* (Colisão): Disparado quando dois objetos interagem fisicamente (sobreposição de suas máscaras de colisão). Esse evento foi utilizado para definir comportamentos como a coleta de pontos, a morte do Pac-Man ao colidir com fantasmas, ou a derrota dos fantasmas durante o efeito dos pontos de poder.
- *Animation End* (Fim da Animação): Controla a transição de sprites (animações) após o término de uma sequência. Foi útil para reiniciar ou alternar animações de movimento e eventos visuais (como a animação de morte do Pac-Man).

Essa estrutura baseada em eventos permitiu uma organização eficiente do comportamento dos objetos, favorecendo a clareza do código ao longo do desenvolvimento.

2.2.1 Objeto *Player*

No evento *Create* do objeto *player*, foram definidas as configurações iniciais do personagem controlado pelo jogador (Pac-Man). Isso inclui a definição do *sprite* padrão, a velocidade da animação, o quadro inicial da animação e a velocidade de movimentação.

Captura de Tela 1 - Evento criar do objeto *player*



Fonte: Autoria Própria no *GameMaker* (2025)

No evento *Etapa*, foi implementada toda a lógica de movimentação do personagem principal, levando em conta as teclas pressionadas, a colisão com obstáculos e o alinhamento com a grade do labirinto. A movimentação só é executada caso o caminho esteja livre (*place_free*) e o personagem esteja perfeitamente alinhado à malha 16x16 do cenário (*place_snapped*), o que garante que ele se mova de forma coerente com o design baseado em células, assim como no jogo original.

Cada tecla de direção (setas do teclado) define a direção do movimento e, quando válida, ativa a movimentação com uma velocidade constante. Além disso, a posição global do *player* é atualizada constantemente para que os fantasmas possam rastreá-lo.

Também foram inseridos mecanismos para controlar a animação: quando o personagem está em movimento, sua animação é ativada com uma velocidade reduzida para garantir fluidez visual; caso contrário, ela é pausada e a imagem é fixada no primeiro quadro. Por fim, o *sprite* exibido é alterado conforme a direção atual, garantindo que o personagem esteja sempre visualmente orientado no sentido do deslocamento (cima, baixo, esquerda ou direita).

Captura de Tela 2 - Evento etapa do objeto *player* (1)

```

obj_player: Etapa
  Etapa
    3 global.pacman_y = y + sprite_yoffset;
    4
    5 if (global.golpe) exit; //se o inimigo atacar todos os comandos serao ignorados
    6
    7 if keyboard_check(vk_right) and place_free(x+1,y) and place_snapped(16,16)
    8 {
    9   direction = 0;
   10   speed = velocidadeMovimento;
   11   global.pacman_dir = 0;//alinhamento em potencia de 4, pois os spr sao 32x32
   12 }
   13
   14 if keyboard_check(vk_left) and place_free(x-1,y) and place_snapped(16,16)
   15 {
   16   direction = 180;
   17   speed = velocidadeMovimento;
   18   global.pacman_dir = 180;//alinhamento em potencia de 4, pois os spr sao 32x32
   19 }
   20
   21 if keyboard_check(vk_up) and place_free(x,y-1) and place_snapped(16,16)
   22 {
   23   direction = 90;
   24   speed = velocidadeMovimento;
   25   global.pacman_dir = 90;//alinhamento em potencia de 4, pois os spr sao 32x32
   26 }
   27
   28 if keyboard_check(vk_down) and place_free(x,y+1) and place_snapped(16,16)
   29 {
   30   direction = 270;
   31   speed = velocidadeMovimento;
   32   global.pacman_dir = 270;//alinhamento em potencia de 4, pois os spr sao 32x32
   33 }

```

Fonte: Autoria Própria no *GameMaker* (2025)

Captura de Tela 3 - Evento etapa do objeto *player* (2)

```

obj_player: Etapa
  Etapa
    34
    35 if speed > 0
    36 {
    37   image_speed = 1;
    38 }
    39
    40 else
    41 {
    42   image_speed = 3;
    43   image_index = 0;
    44 }
    45
    46 switch(direction)
    47 {
    48   case 0:
    49     sprite_index = playerRight;
    50     image_speed = 3;
    51     break;
    52   case 90:
    53     sprite_index = playerUp;
    54     image_speed = 3;
    55     break;
    56   case 180:
    57     sprite_index = playerLeft;
    58     image_speed = 3;
    59     break;
    60   case 270:
    61     sprite_index = PlayerDown;
    62     image_speed = 3;
    63     break;
    64   _}

```

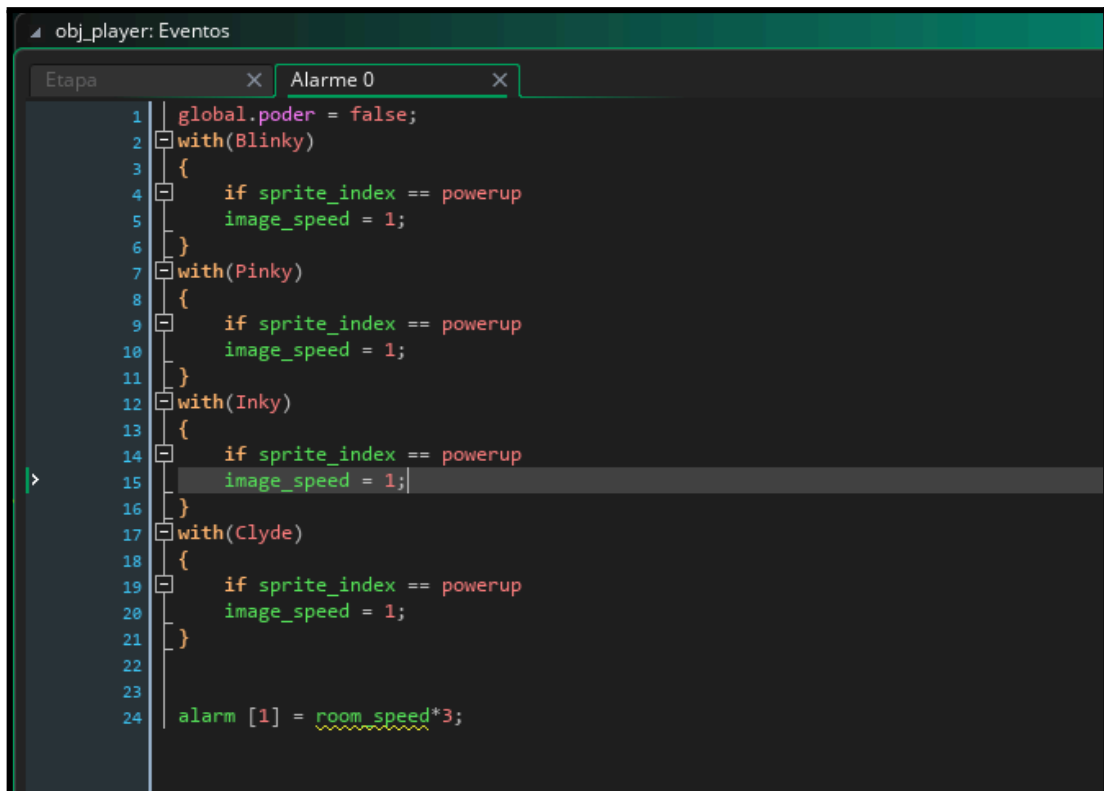
Fonte: Autoria Própria no *GameMaker* (2025)

O evento Alarm 0 é utilizado para controlar o tempo de duração do estado de "poder" que Pac-Man adquire ao consumir um ponto de poder (power pellet).

Quando esse estado termina, a variável global `global.poder` é redefinida para *false*, sinalizando que o *player* perdeu a habilidade temporária de derrotar os inimigos.

Nesse momento, todos os fantasmas (Blinky, Pinky, Inky e Clyde) são verificados. Caso estejam com o sprite correspondente ao estado de vulnerabilidade (*powerup*), suas animações são reativadas ajustando a velocidade (*image_speed = 1*), permitindo a transição visual de volta ao estado normal.

Captura de Tela 4 - Evento alarme 0 do objeto *player*



```

1  global.poder = false;
2  with(Blinky)
3  {
4      if sprite_index == powerup
5      image_speed = 1;
6  }
7  with(Pinky)
8  {
9      if sprite_index == powerup
10     image_speed = 1;
11 }
12 with(Inky)
13 {
14     if sprite_index == powerup
15     image_speed = 1;
16 }
17 with(Clyde)
18 {
19     if sprite_index == powerup
20     image_speed = 1;
21 }
22
23 alarm [1] = room_speed*3;
24

```

Fonte: Autoria Própria no GameMaker (2025)

O evento Alarme 1 é responsável por restaurar os fantasmas ao seu estado original após o término do efeito de poder ativado por Pac-Man. Ele é executado apenas se a variável `global.poder` for falsa, o que garante que os fantasmas só retornem ao comportamento normal quando o efeito realmente tiver se encerrado.

Captura de Tela 5 - Evento alarme 1 do objeto *player*

```

1  if (global.poder) exit;
2  with(Blinky)
3  {
4      image_speed = 0;
5      sprite_index = red;
6      speed = 2;
7      velocidade = 2;
8  }
9
10 with(Pinky)
11 {
12     image_speed = 0;
13     sprite_index = pink;
14     speed = 2;
15     velocidade = 2;
16 }
17
18 with(Inky)
19 {
20     image_speed = 0;
21     sprite_index = blue;
22     speed = 2;
23     velocidade = 2;
24 }
25
26 with(Clyde)
27 {
28     image_speed = 0;
29     sprite_index = orange;
30     speed = 2;
31     velocidade = 2;
32 }
  
```

Fonte: Autoria Própria no *GameMaker* (2025)

O sistema de colisão entre o personagem e os inimigos considera dois cenários principais: com ou sem o efeito de poder ativo (*global.poder*). Quando *player* colide com um fantasma durante o estado de poder (identificado pelo *sprite_index == powerup_2*), ele ganha 200 pontos, um som é reproduzido (*audio_play_sound*), e o fantasma é reiniciado em sua posição inicial (*xstart, ystart*). O sprite do fantasma é alterado para um que indica seu retorno ao labirinto, e sua velocidade padrão é restaurada.

Se o efeito de poder não estiver ativo, e o player colidir com um inimigo estando alinhado horizontal ou verticalmente (*x == other.x or y == other.y*), ele perde uma vida (*global.vidas -= 1*), sua movimentação é pausada (*speed = 0*). Um som de morte é tocado, e um alarme é ativado (*alarm[1]*) para controlar o tempo até a retomada do jogo. A variável *global.golpe* impede que múltiplos eventos de colisão sejam processados ao mesmo tempo.

Captura de Tela 6 - Evento colisão do objeto *player* com o objeto Pinky

```

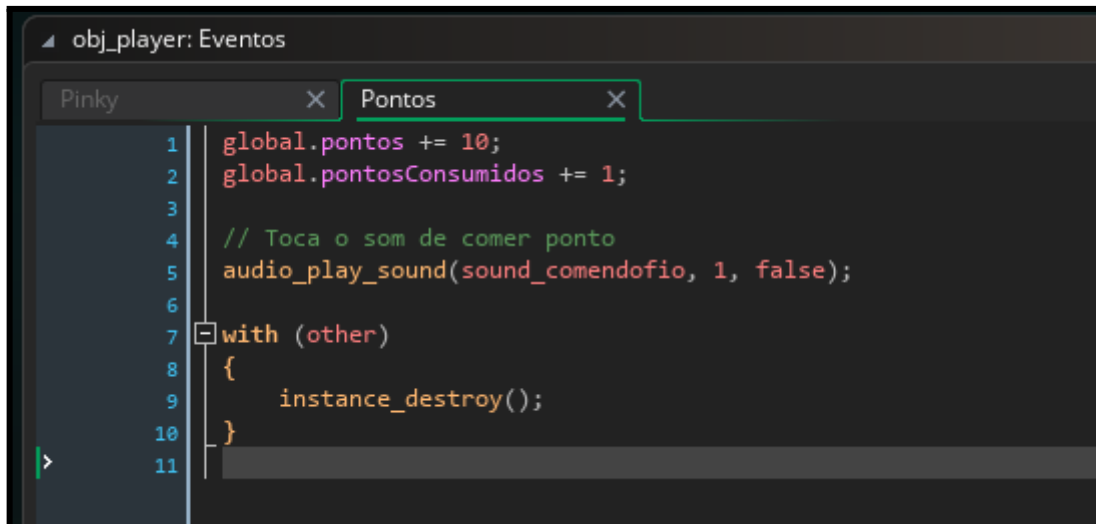
1  if (other.sprite_index == powerup_2)
2  {
3      global.pontos += 200;
4
5      // Toca o som de comer fantasma
6      audio_play_sound(sound_comendofantasma, 1, false);
7
8      with (other)
9      {
10         x = xstart;
11         y = ystart;
12         velocidade = 2;
13         speed = velocidade;
14         sprite_index = PinkyV3;
15     }
16 }
17 else if (!global.golpe) and (x == other.x or y == other.y)
18 {
19     // Toca o som de morte
20     audio_play_sound(sound_morreu, 1, false);
21
22     room_speed = 30;
23     obj_controlador.alarm[1] = room_speed * 15;
24     speed = 0;
25     global.vidas -= 1;
26     sprite_index = powerup_1;
27     image_speed = 1;
28     global.golpe = true;
29 }
30

```

Fonte: Autoria Própria no *GameMaker* (2025)

Quando *player* colide com um ponto comum do mapa, o valor da variável `global.pontos` é incrementado em 10 unidades, e `global.pontosConsumidos` também é atualizado para acompanhar o progresso do jogador. Em seguida, um som é reproduzido com `audio_play_sound` para indicar que o ponto foi coletado.

A instrução `with (other) { instance_destroy(); }` é utilizada para destruir a instância do ponto coletado, removendo-o da tela e impedindo que ele seja coletado novamente.

Captura de Tela 7 - Evento colisão do objeto *player* com o objeto ponto


The screenshot shows the GameMaker IDE interface. At the top, the 'obj_player: Eventos' (obj_player: Events) window is open. It contains two tabs: 'Pinky' and 'Pontos'. The 'Pontos' tab is selected and highlighted with a green border. Below the tabs, a list of events is shown, with line numbers 1 through 11 on the left. The script for the 'Pontos' event is as follows:

```

1 global.pontos += 10;
2 global.pontosConsumidos += 1;
3
4 // Toca o som de comer ponto
5 audio_play_sound(sound_comendofio, 1, false);
6
7 with (other)
8 {
9     instance_destroy();
10 }
11

```

Fonte: Autoria Própria no *GameMaker* (2025)

Quando Pac-Man coleta um ponto de poder, o efeito especial é ativado por meio da variável `global.poder`, que permite que ele elimine fantasmas temporariamente. O sistema adiciona 50 pontos à pontuação total (`global.pontos += 50`) e reproduz um som específico com `audio_play_sound`.

O alarme 0 é ativado (`alarm[0] = room_speed * 5`), iniciando uma contagem que define a duração do efeito de poder. Durante esse período, todos os fantasmas (Blinky, Pinky, Inky e Clyde) têm seus sprites alterados para indicar o estado vulnerável (`sprite_index = powerup_2`).

Captura de Tela 8 - Evento colisão do objeto *player* com o objeto *Power Pellet*

```

1  alarm[0] = room_speed * 5;
2
3  global.pontos += 50;
4  audio_play_sound(sound_comendopowerup, 1, false);
5
6  global.poder = true;
7
8  with (Blinky)
9  {
10     sprite_index = powerup_2;
11     image_speed = 0;
12     image_index = 0;
13     velocidade = 2;
14     speed = 2;
15 }
16
17 with (Pinky)
18 {
19     sprite_index = powerup_2;
20     image_speed = 0;
21     image_index = 0;
22     velocidade = 2;
23     speed = 2;
24 }
25
26 with (Inky)
27 {
28     sprite_index = powerup_2;
29     image_speed = 0;
30     image_index = 0;
31     velocidade = 2;
32     speed = 2;

```

Fonte: Autoria Própria no *GameMaker* (2025)

Quando Pac-Man entra no estado de morte (*sprite* igual a *powerup_1*), inicia-se a reinicialização dos personagens: Pac-Man tem sua direção e animação zeradas, retorna à posição inicial com o *sprite* padrão e a variável de vulnerabilidade (*global.golpe*) é desativada. Os fantasmas Blinky, Pinky, Inky e Clyde também retornam às posições iniciais, com velocidade e *sprites* restaurados, prontos para retomar a perseguição.

Captura de Tela 9 - Evento término da animação do objeto *player*

```

1  if sprite_index == powerup_1
2  {
3      direction = 0;
4      image_speed = 0;
5      x = xstart;
6      y = ystart;
7      sprite_index = playerRight
8      image_index = 0
9
10     global.golpe = false;
11
12     with(Blinky)
13     {
14         x = xstart;
15         y = ystart;
16         velocidade = 2;
17         speed = velocidade;
18         sprite_index = Blinky_v3_1;
19     }
20
21
22     with(Pinky)
23     {
24         x = xstart;
25         y = ystart;
26         velocidade = 2;
27         speed = velocidade;
28         sprite_index = PinkyV3;
29     }

```

Fonte: Autoria Própria no *GameMaker* (2025)

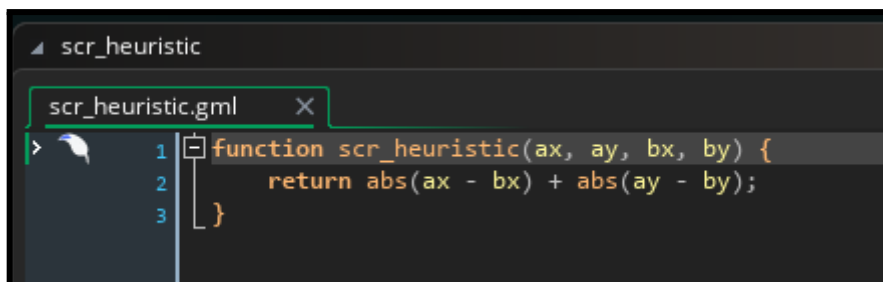
2.2.2 Scripts

Os *scripts* foram desenvolvidos com o objetivo de reutilizar funções aplicadas em múltiplos objetos do jogo. Essa abordagem facilita tanto a manutenção quanto a legibilidade do código, promovendo uma estrutura mais organizada e eficiente durante o processo de desenvolvimento.

2.2.2.1 Função Heurística

A função *heuristic* calcula a distância entre dois pontos (A e B) no mapa, utilizando a distância de Manhattan (soma das diferenças absolutas entre as coordenadas X e Y). Essa abordagem é apropriada para cenários como o do Pac-Man, onde o movimento é feito em uma grade sem diagonais. A heurística fornece uma estimativa do custo para alcançar o objetivo.

Captura de Tela 10 - Script da função heuristic



```

1 function scr_heuristic(ax, ay, bx, by) {
2     return abs(ax - bx) + abs(ay - by);
3 }

```

Fonte: Autoria Própria no GameMaker (2025)

2.2.2.2 Função A estrela

A função *astarpath* implementa o algoritmo de busca A* (A estrela), utilizado para encontrar o caminho mais curto entre dois pontos em um grid, considerando obstáculos e o custo estimado até o destino.

O funcionamento da função envolve várias estruturas auxiliares. A principal delas é a fila de prioridade open, criada com *ds_priority_create*, que armazena os nós (pontos do grid) a serem avaliados, priorizando aqueles com menor custo total estimado (f). Esse custo f é a soma do custo real até o nó (g) mais a estimativa de custo até o destino, calculada pela função *scr_heuristic*, que utiliza a distância de Manhattan (a soma das distâncias horizontal e vertical entre dois pontos).

Além disso, a função usa o mapa closed, criado com *ds_map_create*, para registrar os nós já visitados e evitar reavaliações, e o mapa *came_from* para guardar a origem de cada nó visitado — isso é essencial para reconstruir o caminho final quando o destino for alcançado. Os mapas *g_score* e *f_score* armazenam, respectivamente, o custo real e o custo estimado de cada ponto.

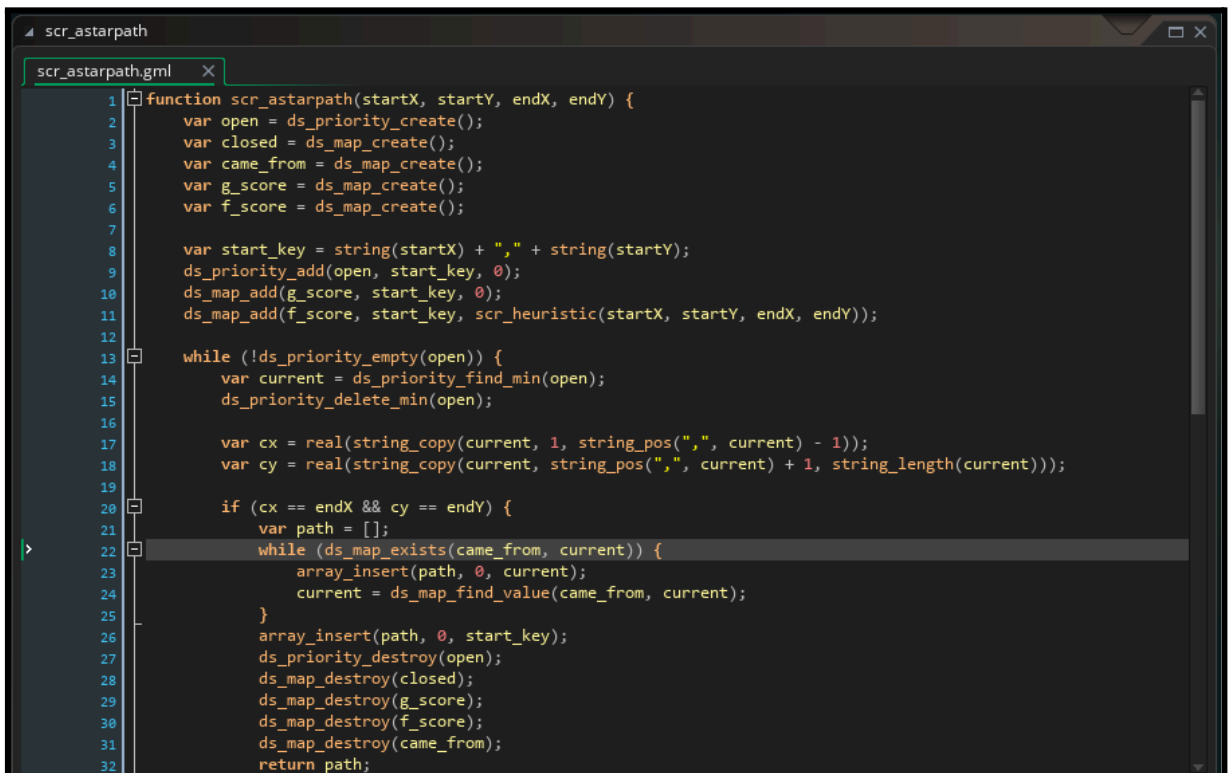
A busca se inicia com o ponto inicial inserido na fila de prioridade usando *ds_priority_add*. Em um laço, o algoritmo remove o nó com menor custo estimado da fila com *ds_priority_find_min* e *ds_priority_delete_min*, e verifica se ele é o destino. Se for, a função reconstrói o caminho completo utilizando o mapa *came_from*, retornando uma lista ordenada de posições.

Caso contrário, o algoritmo avalia os quatro vizinhos do nó atual (cima, baixo, esquerda, direita), ignorando os que já foram visitados ou que estão bloqueados por obstáculos, detectados com a função *place_free*. Para cada vizinho válido, é calculado um custo real temporário (*tentative_g*) e, se esse for o menor já

encontrado, o nó é atualizado nos mapas *came_from*, *g_score* e *f_score*, e reinserido na fila de prioridade com o novo custo total.

Ao final, todas as estruturas auxiliares (*open*, *closed*, *g_score*, *f_score* e *came_from*) são destruídas com suas respectivas funções de destruição, liberando memória. O caminho encontrado é então retornado. Caso não seja possível encontrar um caminho, a função retorna um *array* vazio. Essa abordagem garante eficiência e precisão na navegação por ambientes complexos.

Captura de Tela 11 - Script da função *astarpath* (1)



```

1 function scr_astarpath(startX, startY, endX, endY) {
2     var open = ds_priority_create();
3     var closed = ds_map_create();
4     var came_from = ds_map_create();
5     var g_score = ds_map_create();
6     var f_score = ds_map_create();
7
8     var start_key = string(startX) + "," + string(startY);
9     ds_priority_add(open, start_key, 0);
10    ds_map_add(g_score, start_key, 0);
11    ds_map_add(f_score, start_key, scr_heuristic(startX, startY, endX, endY));
12
13    while (!ds_priority_empty(open)) {
14        var current = ds_priority_find_min(open);
15        ds_priority_delete_min(open);
16
17        var cx = real(string_copy(current, 1, string_pos(",", current) - 1));
18        var cy = real(string_copy(current, string_pos(",", current) + 1, string_length(current)));
19
20        if (cx == endX && cy == endY) {
21            var path = [];
22            while (ds_map_exists(came_from, current)) {
23                array_insert(path, 0, current);
24                current = ds_map_find_value(came_from, current);
25            }
26            array_insert(path, 0, start_key);
27            ds_priority_destroy(open);
28            ds_map_destroy(closed);
29            ds_map_destroy(g_score);
30            ds_map_destroy(f_score);
31            ds_map_destroy(came_from);
32            return path;

```

Fonte: Autoria Própria no *GameMaker* (2025)

Captura de Tela 12 - Script da função *astarpath* (2)

```

scr_astarpath.gml
35 ds_map_add(closed, current, true);
36
37 var dirs = [[1,0], [-1,0], [0,1], [0,-1]];
38 for (var i = 0; i < 4; i++) {
39     var nx = cx + dirs[i][0];
40     var ny = cy + dirs[i][1];
41
42     if (!place_free(nx * 32, ny * 32)) continue;
43
44     var neighbor = string(nx) + "," + string(ny);
45     if (ds_map_exists(closed, neighbor)) continue;
46
47     var tentative_g = ds_map_find_value(g_score, current) + 1;
48
49     if (!ds_map_exists(g_score, neighbor) || tentative_g < ds_map_find_value(g_score, neighbor)) {
50         ds_map_add(came_from, neighbor, current);
51         ds_map_add(g_score, neighbor, tentative_g);
52         var f = tentative_g + scr_heuristic(nx, ny, endX, endY);
53         ds_map_add(f_score, neighbor, f);
54         ds_priority_add(open, neighbor, f);
55     }
56 }
57
58 ds_priority_destroy(open);
59 ds_map_destroy(closed);
60 ds_map_destroy(g_score);
61 ds_map_destroy(f_score);
62 ds_map_destroy(came_from);
63 return [];
64
65

```

Fonte: Autoria Própria no *GameMaker* (2025)

2.2.2.3 Função Busca Gulosa

A função *busca_gulosa* é uma implementação do algoritmo de busca heurística gulosa, utilizada para mover um agente no jogo na direção mais promissora em direção ao jogador (ou outro alvo), com base em uma estimativa de proximidade. Esse tipo de algoritmo é mais simples e rápido que o A*, mas não garante encontrar o caminho mais curto — ele apenas escolhe, a cada passo, o movimento que parece mais vantajoso naquele momento.

Essa função recebe como parâmetros a posição *px*, *py* do alvo (por exemplo, o jogador) e a velocidade com que o agente se move. O agente avalia quatro direções possíveis (cima, baixo, esquerda e direita) e calcula, para cada uma delas, a distância estimada (usando a distância de Manhattan) entre a posição que estaria se seguisse aquela direção e o destino.

Durante esse processo, a função evita retornar para a direção de onde veio (controlado por *last_dx* e *last_dy*), o que evita movimentos oscilatórios. Para cada direção válida (ou seja, sem colisão com uma parede), a função calcula o valor heurístico e armazena as direções com o menor valor encontrado — ou seja, aquelas que mais aproximam o agente do destino.

Caso o agente esteja preso (sem direções que melhorem sua situação), ele considera todas as direções possíveis que não estejam bloqueadas. Por fim, escolhe a melhor direção com base em uma ordem consistente (cima, baixo, esquerda, direita) e retorna os deslocamentos dx e dy multiplicados pela velocidade, que podem ser usados diretamente para definir hspeed e vspeed.

Captura de Tela 13 - Script da função busca gulosa (1)

```

1  function scr_buscagulosa(px, py, velocidade) {
2      var dirs = [
3          [0, -1], // cima
4          [0, 1],  // baixo
5          [-1, 0], // esquerda
6          [1, 0]  // direita
7      ];
8
9      var best_h = 100000;
10     var best_dirs = [];
11
12     var last_dx = (hspeed != 0) ? sign(hspeed / velocidade) : 0;
13     var last_dy = (vspeed != 0) ? sign(vspeed / velocidade) : 0;
14
15     for (var i = 0; i < 4; i++) {
16         var dx = dirs[i][0];
17         var dy = dirs[i][1];
18
19         // Evita voltar
20         if (dx == -last_dx && dy == -last_dy) continue;
21
22         var test_x = x + dx * 32;
23         var test_y = y + dy * 32;
24
25         if (!place_meeting(test_x, test_y, obj_wall)) {
26             var h = abs(test_x - px) + abs(test_y - py);
27             if (h < best_h) {
28                 best_h = h;
29                 best_dirs = [[dx, dy]];
30             } else if (h == best_h) {
31                 array_push(best_dirs, [dx, dy]);
32             }
33         }
34     }
35 }

```

Fonte: Autoria Própria no *GameMaker* (2025)

Captura de Tela 14 - Script da função busca gulosa (2)

```

35
36     // Se estiver preso, libera todas as direções possíveis
37     if (array_length(best_dirs) == 0) {
38         for (var i = 0; i < 4; i++) {
39             var dx = dirs[i][0];
40             var dy = dirs[i][1];
41             var test_x = x + dx * 32;
42             var test_y = y + dy * 32;
43             if (!place_meeting(test_x, test_y, obj_wall)) {
44                 array_push(best_dirs, [dx, dy]);
45             }
46         }
47     }
48
49     // Escolha consistente (prioridade: cima, baixo, esquerda, direita)
50     if (array_length(best_dirs) > 0) {
51         var dx = best_dirs[0][0];
52         var dy = best_dirs[0][1];
53         return [dx * velocidade, dy * velocidade];
54     } else {
55         return [0, 0];
56     }
57 }

```

Fonte: Autoria Própria no *GameMaker* (2025)

2.2.3 Fantasmas

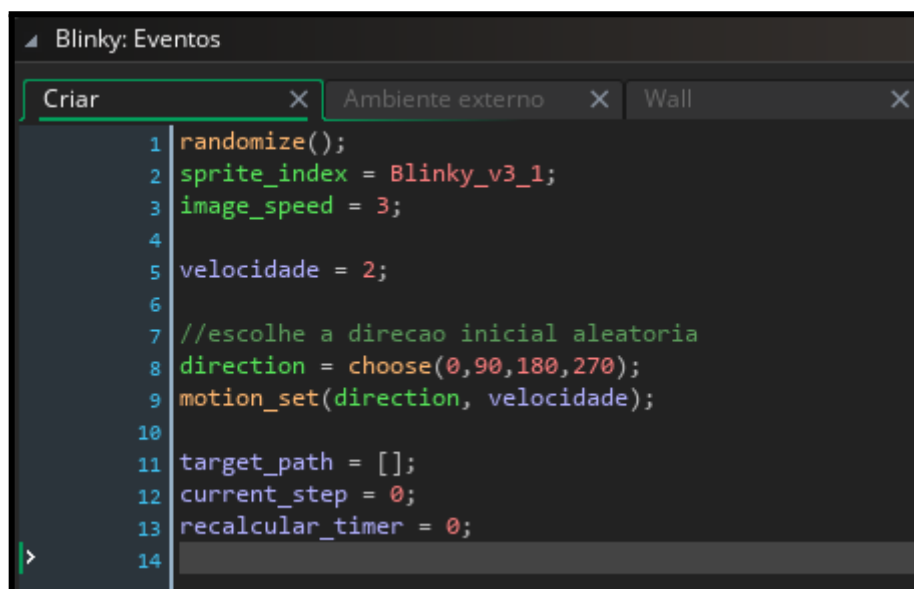
Todos os fantasmas do jogo são criados seguindo uma estrutura padrão, o objeto é inicializado com os mesmos parâmetros básicos, alterando apenas o valor atribuído à variável *sprite_index*, que define qual imagem será exibida para representar o fantasma na tela.

Primeiramente, *randomize()* garante que cada execução do jogo gera resultados aleatórios diferentes. Em seguida, *sprite_index* define a aparência do fantasma, e *image_speed* controla a velocidade da animação do *sprite*. A variável velocidade determina a rapidez com que o fantasma se movimenta. A linha *direction = choose(0,90,180,270)* seleciona aleatoriamente uma das quatro direções cardeais (cima, baixo, esquerda, direita) para o movimento inicial, que é aplicado com *motion_set*.

As variáveis *target_path*, *current_step* e *recalcular_timer* são utilizadas para o controle do caminho a ser seguido pelo fantasma e a frequência com que esse caminho será recalculado, especialmente durante perseguições ao Pac-Man.

Exemplo do Blinky:

Captura de Tela 15 - Exemplo de evento de criação dos objetos dos fantasmas



```
▲ Blinky: Eventos
Criar X Ambiente externo X Wall X
1 randomize();
2 sprite_index = Blinky_v3_1;
3 image_speed = 3;
4
5 velocidade = 2;
6
7 //escolhe a direcao inicial aleatoria
8 direction = choose(0,90,180,270);
9 motion_set(direction, velocidade);
10
11 target_path = [];
12 current_step = 0;
13 recalcular_timer = 0;
14
```

Fonte: Autoria Própria no GameMaker (2025)

No Evento de Etapa do Blinky, o comportamento do fantasma vermelho é implementado de forma a seguir diretamente o Pac-Man, utilizando o algoritmo de busca A* para traçar o caminho mais eficiente até ele.

Logo no início do código, as variáveis globais *global.blinky_x* e *global.blinky_y* são atualizadas com a posição atual do Blinky no jogo, considerando o

deslocamento do *sprite* (útil para outras entidades acompanharem ou evitarem o fantasma).

Em seguida, o código verifica se a variável global `global.golpe` está ativada, indicando que o Pac-Man atingiu Blinky com algum tipo de "poder especial" (como após comer um *power pellet*). Nesse caso, o Blinky é "resetado": sua velocidade é zerada, ele se torna invisível, e sua posição é reiniciada ($x = xstart$, $y = ystart$). Isso simula o retorno do fantasma à sua base após ser derrotado.

Se não estiver sob efeito de golpe, o Blinky permanece visível e continua sua perseguição.

O próximo bloco do código verifica se o Blinky está centralizado na grade (ou seja, perfeitamente alinhado com o *grid* de 32x32 pixels do labirinto). Isso é importante para garantir que ele só mude de direção nos pontos válidos da malha do jogo.

Se estiver centralizado, o código obtém as coordenadas da posição atual e da posição do Pac-Man, e chama a função *astarpath*, que implementa o algoritmo A* para gerar o melhor caminho entre esses dois pontos. O resultado é um *array* com as posições ordenadas que o Blinky deve seguir.

Se o caminho tem mais de uma etapa (ou seja, existe um próximo movimento), o segundo elemento do *array* representa a próxima célula a ser alcançada. O código então extrai essas coordenadas e calcula a diferença (dx e dy) entre a posição atual e o próximo passo. Com esses valores, ele ajusta *hspeed* e *vspeed*, fazendo com que o Blinky se mova diretamente em direção ao *player* com a velocidade definida anteriormente.

Captura de Tela 16 - Evento etapa do objeto blinky (1)

```

1  global.blinky_x = x + sprite_xoffset;
2  global.blinky_y = y + sprite_yoffset;
3
4  if global.golpe {
5      speed = 0;
6      vspeed = 0;
7      hspeed = 0;
8      visible = false;
9
10     x = xstart;
11     y = ystart;
12     exit;
13 } else {
14     visible = true;
15 }
16
17 // Apenas calcula novo caminho se estiver centralizado na grade
18 if place_snapped(32, 32) {
19     var startX = x div 32;
20     var startY = y div 32;
21     var endX = global.pacman_x div 32;
22     var endY = global.pacman_y div 32;
23
24     var path = caminho_a_estrela(startX, startY, endX, endY);
25
26     if (array_length(path) > 1) {
27         // O primeiro elemento é a posição atual, o segundo é o próximo passo
28         var next_step = path[1]; // ex: "10,15"
29         var comma_index = string_pos(",", next_step);
30         var next_x = real(string_copy(next_step, 1, comma_index - 1));
31         var next_y = real(string_copy(next_step, comma_index + 1, string_length(next_step)));
32
33         var dx = next_x - startX;
34         var dy = next_y - startY;
35
36         hspeed = dx * velocidade;
37         vspeed = dy * velocidade;
38     } else {
39         hspeed = 0;
40         vspeed = 0;
41     }
42
43     //sprite_index = Blinky_v2;
44 }

```

Fonte: Autoria Própria no *GameMaker* (2025)

Captura de Tela 17 - Evento etapa do objeto blinky (2)

```

33     var dx = next_x - startX;
34     var dy = next_y - startY;
35
36     hspeed = dx * velocidade;
37     vspeed = dy * velocidade;
38 } else {
39     hspeed = 0;
40     vspeed = 0;
41 }
42
43 //sprite_index = Blinky_v2;
44 }
45

```

Fonte: Autoria Própria no *GameMaker* (2025)

No Evento de Etapa do Pinky, o comportamento do fantasma rosa é programado para tentar antecipar os movimentos do jogador, buscando se posicionar à frente dele. Para isso, utiliza-se a função *busca_gulosa*, que implementa um algoritmo de busca heurística simples, direcionando Pinky até um ponto-alvo baseado na direção atual do jogador.

Logo no início do código, é verificado se as variáveis globais *global.pacman_x* e *global.pacman_y* existem, o que garante que a posição atual do *player* está

disponível para cálculo. Em seguida, é feita a verificação *place_snapped(32, 32)*, que assegura que o Pinky esteja centralizado em um bloco da grade 32x32 antes de tomar uma nova decisão de movimento. Esse alinhamento com a malha do labirinto é fundamental para garantir movimentos precisos e válidos.

Dentro desse bloco, o código distingue dois comportamentos diferentes com base no estado do jogo. Se a variável *global.poder* estiver ativada (indicando que o Pac-Man está sob o efeito de um power pellet), Pinky entra em modo de fuga. Nesse estado, o fantasma calcula um ponto na direção oposta à que o Pac-Man está seguindo, utilizando um deslocamento de 128 *pixels* (equivalente a quatro blocos). Por exemplo, se o jogador estiver indo para cima, Pinky tentará fugir 128 *pixels* para baixo.

Caso contrário, no modo padrão, Pinky calcula um alvo 128 pixels à frente da direção atual do Pac-Man, tentando prever para onde o jogador irá.

Com o alvo definido (*target_x* e *target_y*), a função *busca_gulosa* é chamada para traçar a direção ideal até o destino. O retorno dessa função é armazenado na variável *direcao*, que contém os valores de *hspeed* e *vspeed* correspondentes às velocidades horizontal e vertical do Pinky. Esses valores são então aplicados para movimentar o fantasma na direção desejada.

Captura de Tela 18 - Evento etapa do objeto pinky

```

1  if variable_global_exists("pacman_x") && variable_global_exists("pacman_y")
2  {
3      if place_snapped(32, 32) {
4          var target_x, target_y;
5
6          if (global.poder) {
7              // Fugir do Pac-Man - calcula um ponto na direção oposta
8              var offset = 128; // 4 blocos de distância
9              var dir = global.pacman_dir;
10             target_x = global.pacman_x;
11             target_y = global.pacman_y;
12
13             switch (dir) {
14                 case 0: target_x -= offset; break; // Pac-Man indo para a direita + fugir para esquerda
15                 case 180: target_x += offset; break; // Pac-Man indo para a esquerda + fugir para direita
16                 case 90: target_y += offset; break; // Pac-Man indo para cima + fugir para baixo
17                 case 270: target_y -= offset; break; // Pac-Man indo para baixo + fugir para cima
18             }
19         } else {
20             // Comportamento padrão - perseguição
21             target_x = global.pacman_x;
22             target_y = global.pacman_y;
23
24             var offset = 128;
25             switch (global.pacman_dir) {
26                 case 0: target_x += offset; break;
27                 case 180: target_x -= offset; break;
28                 case 90: target_y -= offset; break;
29                 case 270: target_y += offset; break;
30             }
31         }
32
33         // Executa a busca gulosa com base no target ajustado
34         var direcao = busca_gulosa(target_x, target_y, velocidade);
35         hspeed = direcao[0];
36         vspeed = direcao[1];
37     }
38 }
  
```

Fonte: Autoria Própria no *GameMaker* (2025)

No Evento de Etapa do Clyde, o fantasma azul apresenta um comportamento híbrido entre perseguição e evasão. Ele utiliza o algoritmo A* (implementado na função *astarath*) para seguir o Pac-Man de forma semelhante ao Blinky, mas com uma particularidade: quando se aproxima demais do jogador, ele abandona a perseguição e recua para sua posição inicial no labirinto.

Em seguida, o código verifica se a variável *global.golpe* está ativada. Isso representa o estado em que Clyde foi atingido por um poder especial do Pac-Man. Nesse caso, o fantasma é "resetado": sua velocidade é zerada (*hspeed*, *vspeed* e *speed*), ele se torna invisível (*visible = false*) e retorna à sua posição inicial (*xstart*, *ystart*), simulando o retorno à base. O comando *exit* garante que nenhuma outra lógica seja executada neste ciclo de atualização.

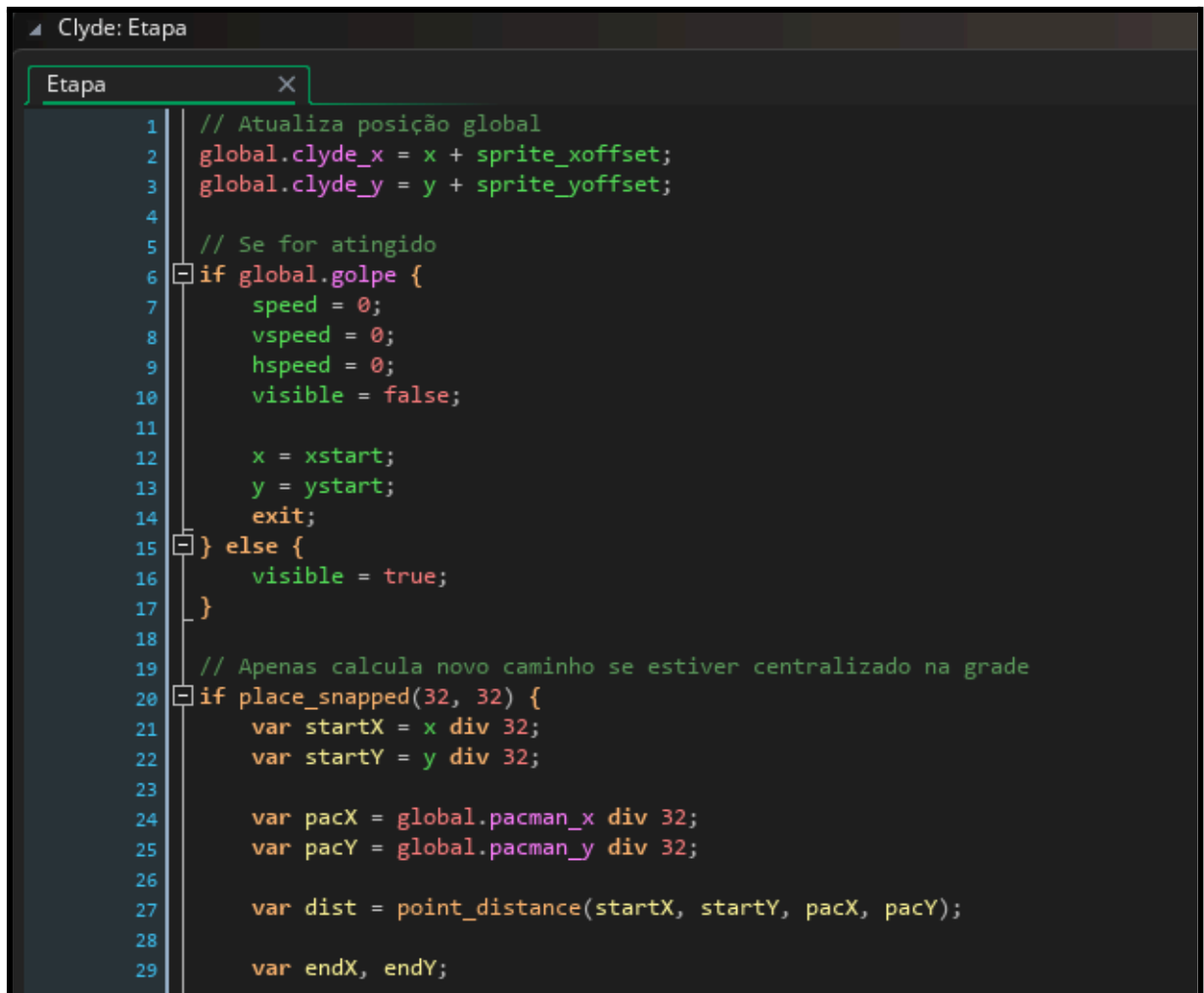
Caso não esteja sob efeito de golpe, Clyde permanece visível e continua seu comportamento normal. A próxima etapa é verificar se o fantasma está centralizado

na grade de 32x32 *pixels* com a função *place_snapped*(32, 32), garantindo que decisões de rota sejam feitas apenas nos cruzamentos válidos do labirinto.

Se centralizado, o código calcula a distância entre Clyde e o Pac-Man utilizando *point_distance*, com coordenadas normalizadas pela grade. Se essa distância for inferior a quatro blocos (ou seja, menor que 128 *pixels*), o fantasma interpreta isso como uma proximidade perigosa e decide recuar, definindo como destino sua posição inicial. Caso contrário, ele define o Pac-Man como alvo e continua a perseguição.

Em ambos os casos, o algoritmo A* é utilizado para gerar um caminho entre a posição atual e o destino (*startX, startY* → *endX, endY*). Se o caminho tiver mais de um passo, o segundo ponto da rota (índice 1) é extraído e convertido em coordenadas numéricas. A diferença entre o ponto atual e o próximo (*dx, dy*) é então usada para ajustar *hspeed* e *vspeed*, aplicando a velocidade do Clyde em direção ao próximo nó do caminho.

Captura de Tela 19 - Evento etapa do objeto clyde



```
1 // Atualiza posição global
2 global.clyde_x = x + sprite_xoffset;
3 global.clyde_y = y + sprite_yoffset;
4
5 // Se for atingido
6 if global.golpe {
7     speed = 0;
8     vspeed = 0;
9     hspeed = 0;
10    visible = false;
11
12    x = xstart;
13    y = ystart;
14    exit;
15 } else {
16     visible = true;
17 }
18
19 // Apenas calcula novo caminho se estiver centralizado na grade
20 if place_snapped(32, 32) {
21     var startX = x div 32;
22     var startY = y div 32;
23
24     var pacX = global.pacman_x div 32;
25     var pacY = global.pacman_y div 32;
26
27     var dist = point_distance(startX, startY, pacX, pacY);
28
29     var endX, endY;
```

Fonte: Autoria Própria no *GameMaker* (2025)

Captura de Tela 20 - Evento etapa do objeto clyde (2)

```

30
31 // Se estiver perto demais do Pac-Man, foge para o ponto inicial
32 if (dist < 4) {
33     endX = xstart div 32;
34     endY = ystart div 32;
35 } else {
36     // Caso contrário, persegue o Pac-Man
37     endX = pacX;
38     endY = pacY;
39 }
40
41 var path = caminho_a_estrela(startX, startY, endX, endY);
42
43 if (array_length(path) > 1) {
44     var next_step = path[1]; // ex: "10,15"
45     var comma_index = string_pos(",", next_step);
46     var next_x = real(string_copy(next_step, 1, comma_index - 1));
47     var next_y = real(string_copy(next_step, comma_index + 1, string_length(next_step)));
48
49     var dx = next_x - startX;
50     var dy = next_y - startY;
51
52     hspeed = dx * velocidade;
53     vspeed = dy * velocidade;
54 } else {
55     hspeed = 0;
56     vspeed = 0;
57 }
58 }
59

```

Fonte: Autoria Própria no *GameMaker* (2025)

No Evento de Etapa do Inky, o comportamento do fantasma azul-claro é definido por uma lógica mais complexa e imprevisível, combinando estratégias dos fantasmas Blinky e Pinky. Em vez de simplesmente perseguir o player diretamente, Inky calcula sua rota com base em uma projeção à frente do jogador e na posição atual de Blinky, o que resulta em movimentos menos previsíveis para o jogador.

A execução do comportamento começa com uma verificação da existência das variáveis globais necessárias: `pacman_x`, `pacman_y`, `blinky_x` e `blinky_y`. Essas variáveis representam, respectivamente, as posições do Pac-Man e do Blinky no labirinto. Se todas estiverem disponíveis, e o Inky estiver centralizado na grade (verificado por `place_snapped(32, 32)`), o código segue com os cálculos.

Para evitar *loops* indesejados (em que o fantasma possa ficar preso repetindo o mesmo movimento), é realizada uma verificação adicional: o código compara a posição atual do Inky com a última posição visitada, armazenada nas variáveis `global.last_inky_tile_x` e `global.last_inky_tile_y`. Se ele estiver no mesmo *tile* da última verificação, o movimento é interrompido nesse ciclo com `exit`.

Em seguida, o código prevê uma posição à frente do jogador, deslocando suas coordenadas atuais em 64 pixels (dois blocos) na direção em que o

personagem está se movendo. Esse ponto representa uma antecipação da trajetória de Pac-Man, como faz Pinky.

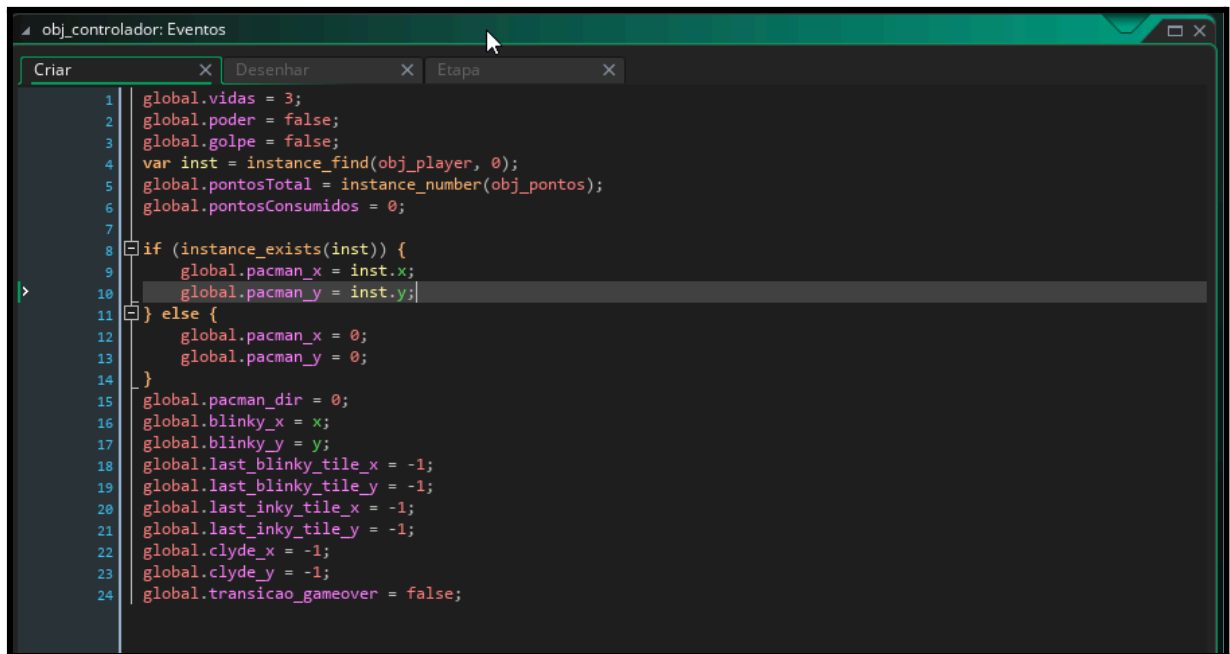
Com essa posição projetada (px , py), o código então calcula um vetor que vai da posição de Blinky até o ponto previsto, e o dobra em magnitude. A soma desse vetor à posição de Blinky define o target de Inky, ou seja, a célula do labirinto que ele tentará alcançar. Essa lógica faz com que Inky às vezes ataque de ângulos inesperados, criando situações de cerco em conjunto com Blinky.

Por fim, o código utiliza a função `busca_gulosa`, que implementa uma estratégia de busca heurística, para determinar a melhor direção a seguir em direção ao ponto-alvo calculado. O resultado dessa função é um vetor de movimento ($hspeed$ e $vspeed$), que é aplicado diretamente ao fantasma para movimentá-lo no labirinto.

2.2.4 Controlador

O objeto controlador atua como gerenciador da lógica das fases, sendo responsável por gerenciar variáveis globais, coordenar o estado geral da partida e exibir informações ao jogador, como vidas restantes e pontuação acumulada em cada fase. A seguir, é apresentada a lógica contida no evento de criação responsável pela inicialização desses elementos:

No evento *Create* do objeto controlador, são inicializadas as variáveis globais que mantêm o estado geral do jogo. Isso inclui a quantidade de vidas do jogador (`global.vidas`), o status de ações como o modo de poder (`global.poder`) e a variável booleana que confere a colisão entre o personagem e os inimigos (`global.golpe`). Além disso, o código localiza o jogador e armazena sua posição inicial nas variáveis globais `global.pacman_x` e `global.pacman_y`. Também é definido o número total de pontos disponíveis no cenário (`global.pontosTotal`) e os pontos já consumidos (`global.pontosConsumidos`). Por fim, o código registra as posições iniciais dos fantasmas e variáveis auxiliares utilizadas na lógica de movimentação de Blinky, Inky e Clyde, além de preparar o controle para a tela de transição em caso de Game Over.



```

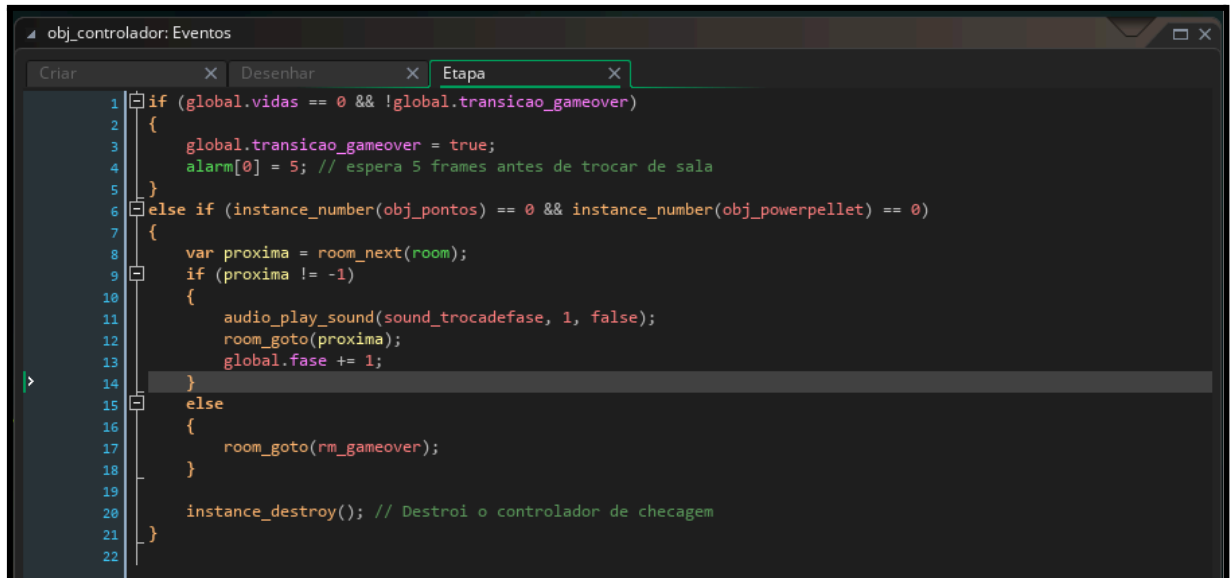
1  global.vidas = 3;
2  global.poder = false;
3  global.golpe = false;
4  var inst = instance_find(obj_player, 0);
5  global.pontosTotal = instance_number(obj_pontos);
6  global.pontosConsumidos = 0;
7
8  if (instance_exists(inst)) {
9      global.pacman_x = inst.x;
10     global.pacman_y = inst.y;
11 } else {
12     global.pacman_x = 0;
13     global.pacman_y = 0;
14 }
15 global.pacman_dir = 0;
16 global.blinky_x = x;
17 global.blinky_y = y;
18 global.last_blinky_tile_x = -1;
19 global.last_blinky_tile_y = -1;
20 global.last_inky_tile_x = -1;
21 global.last_inky_tile_y = -1;
22 global.clyde_x = -1;
23 global.clyde_y = -1;
24 global.transicao_gameover = false;

```

Fonte: Autoria Própria no GameMaker (2025)

No evento *Step* do objeto controlador, é realizada a verificação contínua do estado do jogo para determinar se o jogador perdeu todas as vidas ou concluiu uma fase. Primeiramente, o código checa se o número de vidas (`global.vidas`) chegou a zero e se a transição para a tela de *game over* ainda não foi iniciada. Caso ambas as condições sejam verdadeiras, a variável `global.transicao_gameover` é ativada para evitar repetições, e um temporizador (`alarm[0]`) é definido para aguardar 5 frames antes de realizar a transição de sala.

Em seguida, o código verifica se todos os pontos normais (`obj_pontos`) e os pontos especiais (`obj_powerpellet`) do cenário foram consumidos. Se isso ocorrer, significa que o jogador completou a fase. Nesse caso, a função `room_next()` busca a próxima sala disponível. Se uma próxima fase existir, ela é carregada por meio da função `room_goto()`, acompanhada da reprodução de um som de transição e do incremento na variável `global.fase`. Caso contrário, o jogo é encerrado e direcionado para a sala de *gameover* (`rm_gameover`). Após qualquer uma dessas ações, o controlador é destruído com `instance_destroy()` para evitar redundância nas checagens.



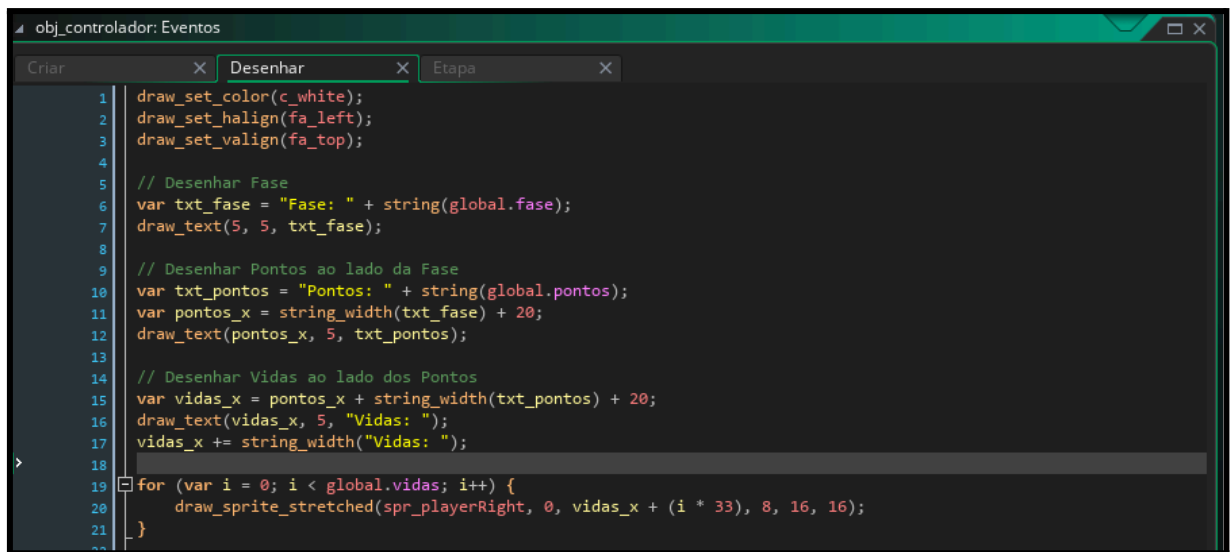
Fonte: Autoria Própria no *GameMaker* (2025)

O evento Desenhar do objeto controlador é responsável por exibir na tela informações importantes do jogo de forma clara e organizada para o jogador. Inicialmente, são definidas a cor branca e os alinhamentos horizontal e vertical para o texto, garantindo uma visualização padronizada no canto superior esquerdo da tela.

Em seguida, são desenhados três elementos principais:

- Fase atual: é exibida a *string* “Fase: ” seguida do número correspondente à fase, posicionado no canto superior esquerdo.
- Pontuação: logo ao lado da fase, é mostrado o texto “Pontos: ” junto da pontuação acumulada do jogador (`global.pontos`). A posição é calculada dinamicamente com base na largura do texto da fase, garantindo espaçamento adequado.
- Vidas restantes: por fim, o número de vidas é exibido após a pontuação. A palavra “Vidas: ” é desenhada, seguida de um ícone do personagem principal (`spr_playerRight`) repetido de acordo com a quantidade de vidas em `global.vidas`. Cada ícone é posicionado com um espaçamento constante, simulando um contador visual de vidas.

Captura de Tela 23 - Evento desenhar do objeto controlador



```

1 draw_set_color(c_white);
2 draw_set_halign(fa_left);
3 draw_set_valign(fa_top);
4
5 // Desenhar Fase
6 var txt_fase = "Fase: " + string(global.fase);
7 draw_text(5, 5, txt_fase);
8
9 // Desenhar Pontos ao lado da Fase
10 var txt_pontos = "Pontos: " + string(global.pontos);
11 var pontos_x = string_width(txt_fase) + 20;
12 draw_text(pontos_x, 5, txt_pontos);
13
14 // Desenhar Vidas ao lado dos Pontos
15 var vidas_x = pontos_x + string_width(txt_pontos) + 20;
16 draw_text(vidas_x, 5, "Vidas: ");
17 vidas_x += string_width("Vidas: ");
18
19 for (var i = 0; i < global.vidas; i++) {
20     draw_sprite_stretched(spr_playerRight, 0, vidas_x + (i * 33), 8, 16, 16);
21 }

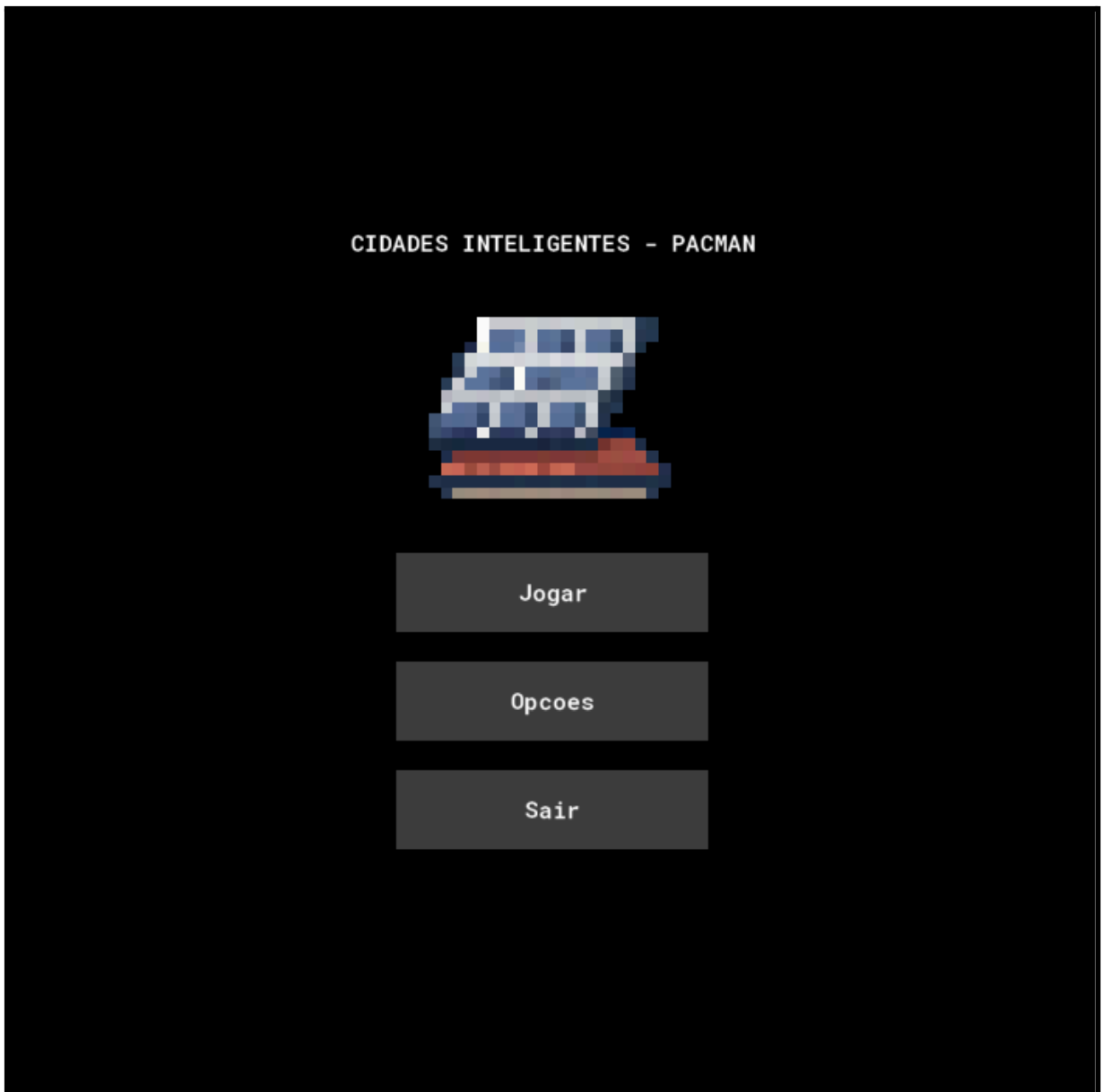
```

Fonte: Autoria Própria no *GameMaker* (2025)

2.2.1 Telas

Após apresentar o funcionamento do jogo, vamos detalhar agora cada uma das telas disponíveis. A seguir, uma breve descrição das funcionalidades presentes em cada uma delas.

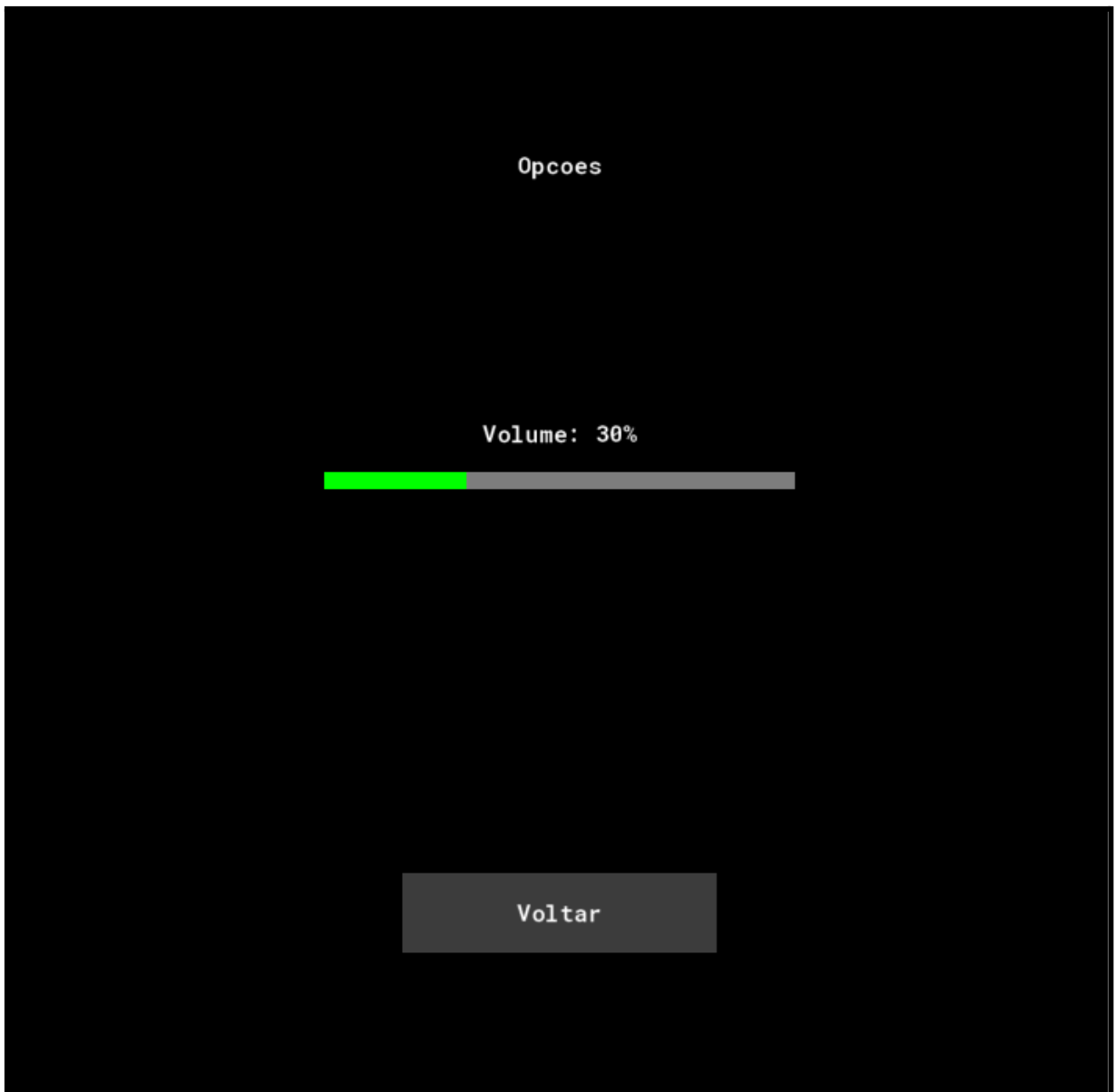
A tela inicial do jogo é simples e intuitiva, exibindo três opções principais para o jogador. O botão "Jogar" direciona imediatamente para a tela de instruções, onde o jogador pode entender as regras antes de começar a partida. Já no botão "Opções", é possível ajustar o volume do áudio do jogo. Por fim, o botão "Sair" encerra o jogo automaticamente quando selecionado, fechando a aplicação.

Captura de Tela 24 - Tela Inicial

Fonte: Autoria Própria no *GameMaker* (2025)

Na tela de opções, o jogador encontra um controle deslizante que permite ajustar o volume do jogo. Além dessa configuração, um botão de "Voltar" permite retornar ao menu inicial.

Captura de Tela 25 -Tela de Opções



Fonte: Autoria Própria no *GameMaker* (2025)

A tela de instruções apresenta todas as informações que o jogador precisa para começar a jogar. Nela, são explicados os objetivos do jogo, os comandos disponíveis e dicas essenciais para jogabilidade.

Além disso, um botão "Iniciar" é exibido de forma destacada, permitindo que, após ler as instruções, o jogador possa avançar diretamente para a primeira fase do jogo.

Captura de Tela 26 - Tela de Instruções Iniciais

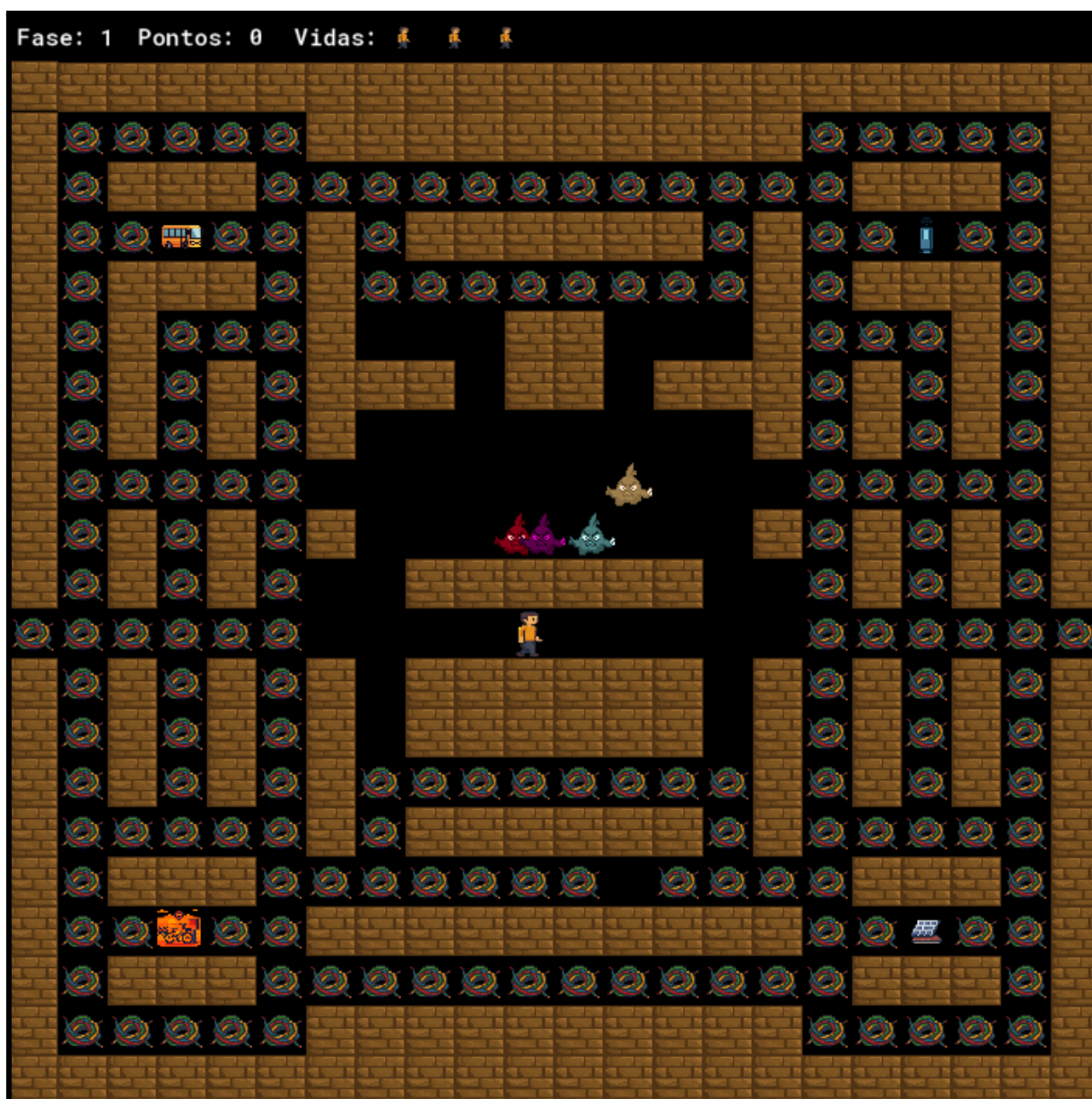


Fonte: Autoria Própria no *GameMaker* (2025)

Esta é a tela inicial da primeira fase, onde o jogador encontra todas as informações essenciais dispostas na parte superior da tela. A interface mostra claramente em qual fase o usuário se encontra, exibe a pontuação acumulada durante a partida e indica quantas vidas ainda restam.

A progressão do jogo segue um fluxo definido: ao completar com sucesso esta primeira fase, o jogador avança automaticamente para a segunda fase. Por outro lado, caso o jogador perca todas as suas vidas, será direcionado imediatamente para a tela de *gameover*.

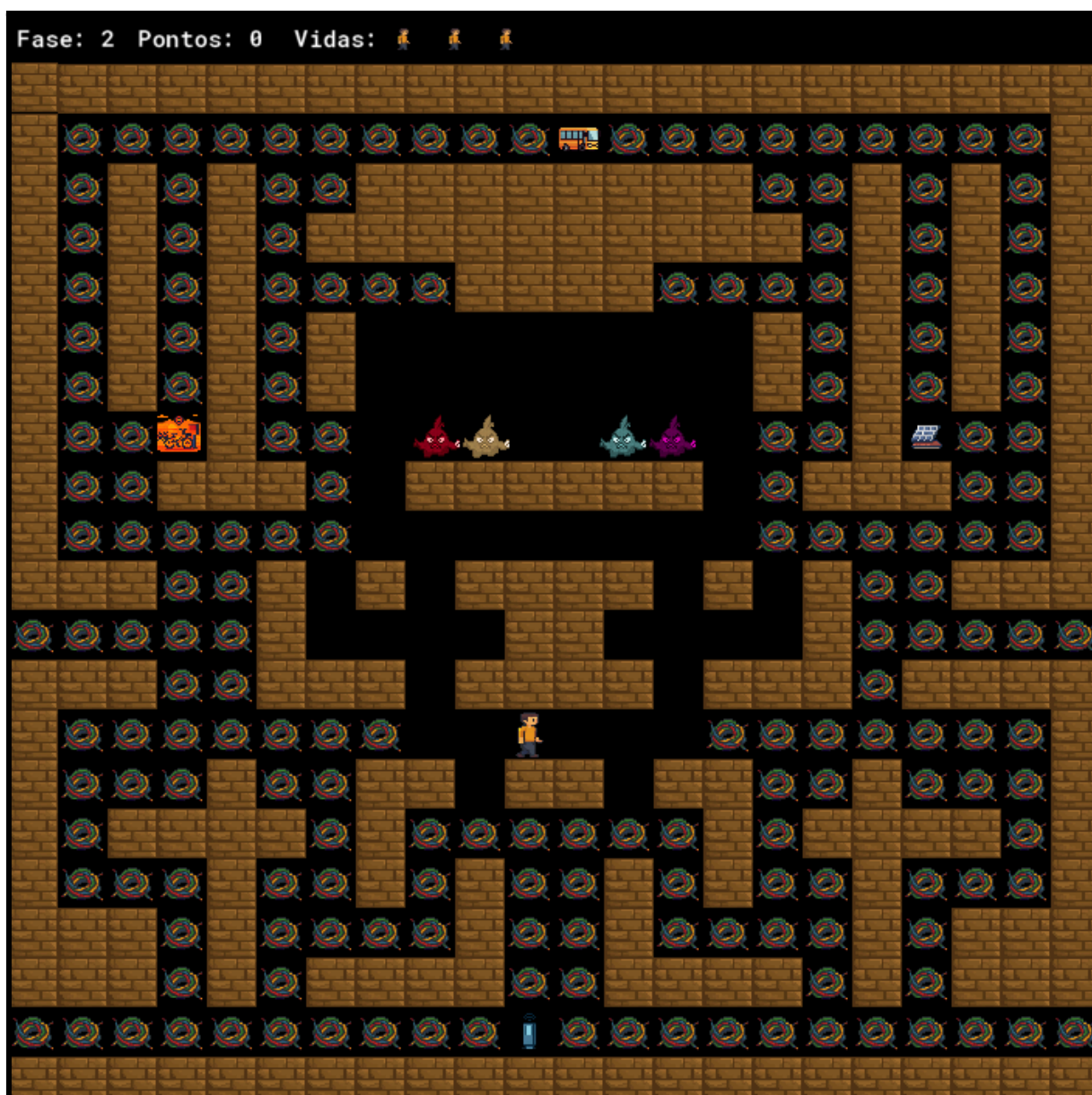
Captura de Tela 27 - Primeira Fase



Fonte: Autoria Própria no *GameMaker* (2025)

Esta fase funciona igual à primeira: mostra no topo da tela o número da fase, os pontos e as vidas. Se passar, vai pra próxima fase. Se perder todas as vidas, vai direto pro *gameover*.

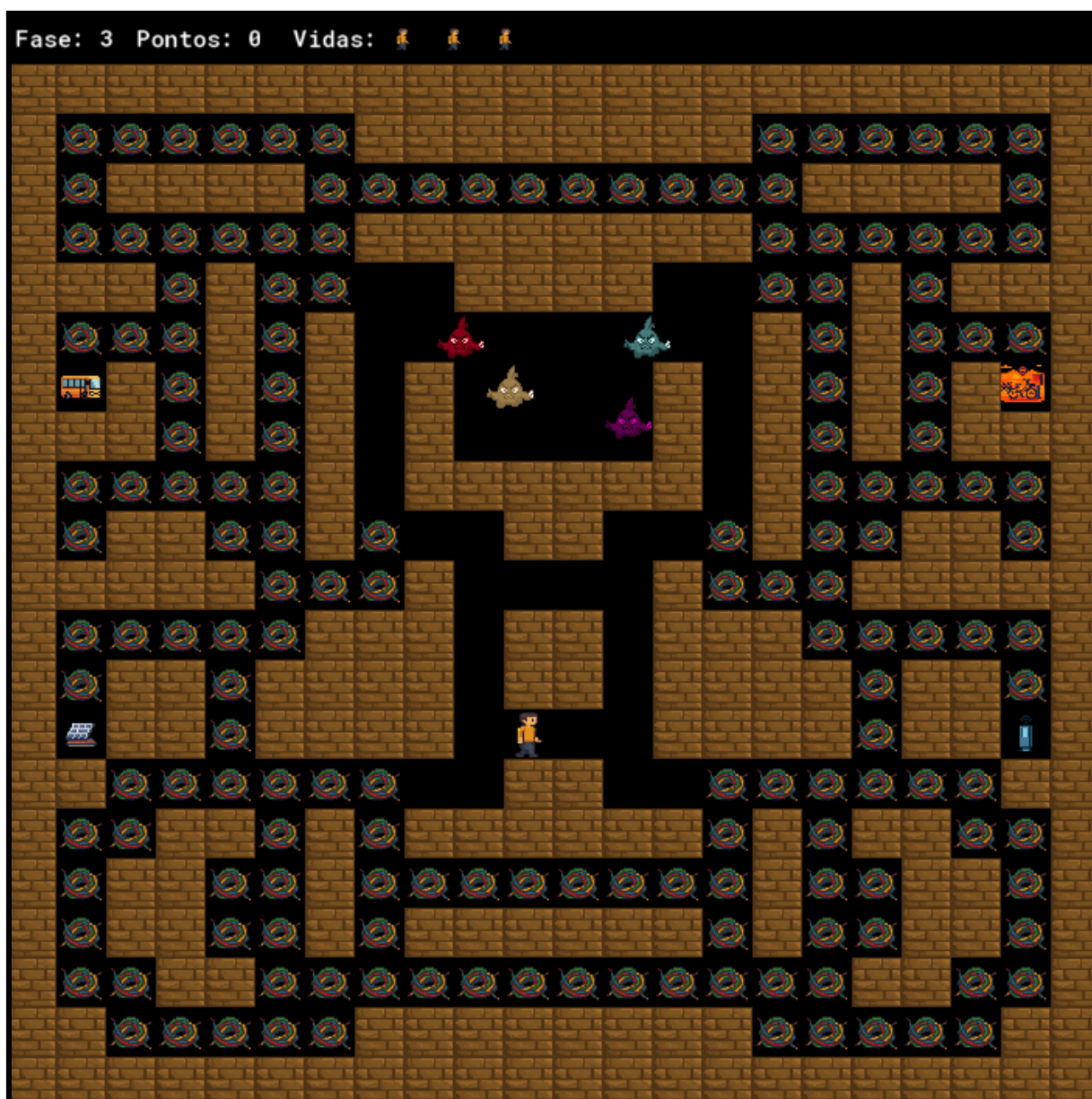
Captura de Tela 28 - Segunda Fase



Fonte: Autoria Própria no *GameMaker* (2025)

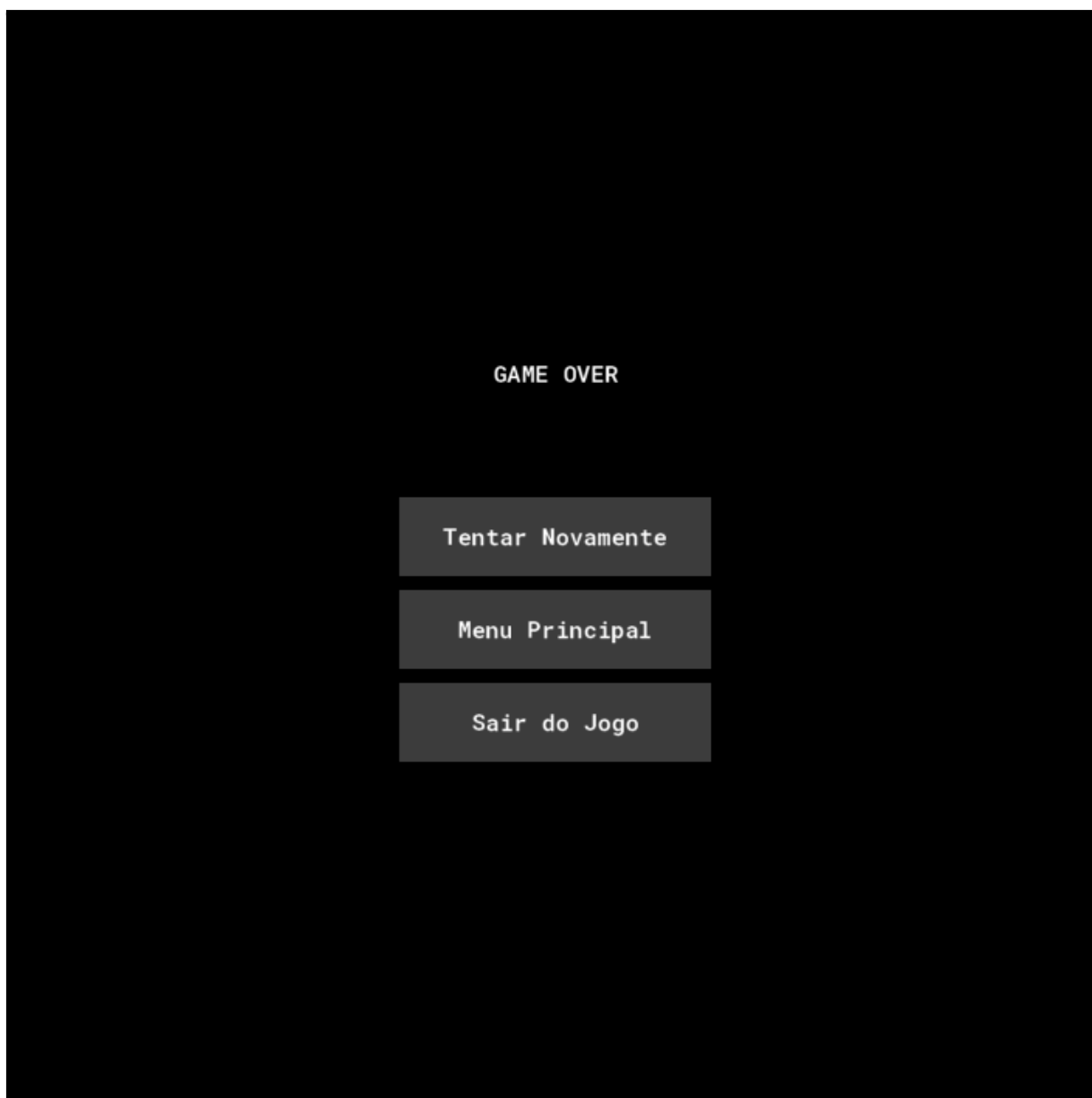
Assim como nas fases anteriores, esta tela mantém os mesmos elementos básicos, a única diferença ocorre ao concluir a fase - em vez de avançar para uma próxima etapa, o jogador é levado diretamente para a tela de fim de jogo, marcando o final da experiência. Caso perca todas as vidas durante a fase, como antes, será direcionado para a tela de *gameover*.

Captura de Tela 29 - Terceira Fase



Fonte: Autoria Própria no *GameMaker* (2025)

Esta é a tela de *gameover* que aparece quando o jogador perde todas as vidas. Ela oferece três opções simples: o botão "Tentar Novamente" reinicia o jogo na primeira fase, "Menu Principal" leva de volta à tela inicial e "Sair" fecha o jogo completamente.

Captura de Tela 30 - Tela de *Gameover*

Fonte: Autoria Própria no *GameMaker* (2025)

Esta é a tela final que aparece quando o jogador completa todas as três fases. Ela parabeniza o jogador pela conquista e oferece duas opções: voltar para a tela inicial ou sair do jogo.

Captura de Tela 31 - Tela Final



Fonte: Autoria Própria no *GameMaker* (2025)

2.3 Resultado

Conforme demonstrado na implementação e metodologia adotada, os objetivos deste trabalho foram plenamente alcançados com o desenvolvimento de um jogo no estilo *Pac-Man*, adaptado ao tema de cidades inteligentes. O produto atende os requisitos inicialmente propostos, apresentando:

Funcionalidade Completa: todas as mecânicas essenciais foram implementadas, incluindo:

- Sistema de movimentação fluida do personagem principal;

- Comportamento inteligente dos inimigos com diferentes estratégias de perseguição;
- Sistema de coleta de itens e pontuação;
- Telas de menu, pausa e game over totalmente funcionais.

Adaptação Temática: a transposição do conceito clássico para o contexto de cidades inteligentes foi realizada através de:

- Elementos visuais que representam tecnologias urbanas;
- Mecânicas de jogo que simulam desafios de gestão urbana;
- Narrativa integrada ao tema proposto.

Como comprovação dos resultados, foi produzido e disponibilizado um vídeo demonstrativo abrangente, capturado utilizando o software *Open Broadcaster Software Studio (OBS Studio)*, ferramenta essencial para a gravação do material apresentado. O vídeo documenta:

- Demonstração completa de todas as telas do sistema (menu inicial, configurações, gameplay e telas de fim);
- Funcionamento da inteligência artificial dos agentes inimigos;
- Todas as mecânicas de jogo em operação;
- Transições fluidas entre os diferentes estados do jogo.

Vídeo de demonstração: <https://youtu.be/3AsMSOBTlyY>

A gravação, realizada com o *OBS Studio*, atesta o cumprimento dos requisitos funcionais e também a estabilidade e performance do jogo desenvolvido, servindo como evidência do sucesso da implementação.

3 CONCLUSÃO

Este trabalho demonstrou com sucesso a aplicação prática de técnicas de Inteligência Artificial no desenvolvimento de uma versão adaptada do clássico *Pac-Man*. Através da modelagem do ambiente como um espaço de estados e da implementação dos algoritmos A* e busca gulosa com heurísticas específicas, foi possível criar agentes adversários (fantasmas) com comportamentos distintos e estrategicamente desafiadores. Destaca-se que o desenvolvimento contou com o apoio fundamental de ferramentas de IA generativa (ChatGPT e DeepSeek) na implementação de partes do código, criação de assets visuais e elaboração de trechos da documentação.

A plataforma *GameMaker* mostrou-se adequada para a implementação, permitindo a integração eficiente entre os componentes desenvolvidos manualmente

e aqueles criados com assistência de IA. A estrutura de três níveis com labirintos distintos comprovou a flexibilidade da solução, com os fantasmas adaptando suas estratégias em tempo real. O uso combinado de técnicas tradicionais e assistência por inteligência artificial revelou-se eficaz na solução dos desafios.

Os resultados alcançados validam plenamente a abordagem proposta: os fantasmas apresentaram comportamentos coerentes com suas personalidades originais (Blinky agressivo, Pinky preditiva, Inky imprevisível e Clyde errático), agora implementados através de técnicas formais de IA. O vídeo demonstrativo atesta o funcionamento integral do sistema, fruto desta colaboração entre desenvolvimento humano e assistência computacional inteligente.

Este projeto serviu como ponte entre teoria e prática, além de demonstrar o potencial das ferramentas de IA generativa como copilotos no desenvolvimento de jogos. A metodologia empregada mostrou-se promissora para futuras expansões, podendo ser aplicada à incorporação de técnicas mais avançadas ou à criação de conteúdos adicionais.

REFERÊNCIAS

- OPENAI. **ChatGPT - Inteligência Artificial**. 16 abr. 2025.
Disponível em: <https://chatgpt.com/>. Acesso em: 1 maio 2025.
- DEEPSEEK. **DeepSeek - Inteligência Artificial**. 5 set. 2024.
Disponível em: <https://www.deepseek.com/>. Acesso em: 1 maio 2025.
- OBS Project. **OBS Studio - Free and open source software for live streaming and screen recording**. 30 de abr. de 2024.
Disponível em: <https://obsproject.com/>. Acesso em: 10 maio 2025.
- YoYo Games Ltd. GameMaker Manual. **GameMaker**. 10 maio 2025.
Disponível em: <https://manual.gamemaker.io/monthly/en/#t=Content.htm>. Acesso em: 10 maio 2025.
- SWARD, Bradley. **Pac-Man With Arcade Ghost AI Using GML In GameMaker Studio 2**. 7 nov 2020.
Disponível em: <https://www.youtube.com/watch?v=z9oVSM40N1I>. Acesso em: 10 maio 2025.
- DIAS, Bianca. **Inteligência Artificial - Trabalho 1**. [s.d.]*
Disponível em: <http://www2.ic.uff.br/~bianca/ia-pos/t1.html>. Acesso em: 10 maio 2025.
- Moreira, Flávia. **Explorando a Busca Gulosa em Inteligência Artificial**. 30 de maio de 2024.
Disponível em: <https://evolvers.com.br/explorando-busca-gulosa/>. Acesso em: 10 maio 2025.
- LUIZ, Gabriel. **Descobrindo a eficácia da busca heurística**. 5 jan. 2024.
Disponível em:
<https://medium.com/@gabrielluizone/descobrindo-a-eficácia-da-busca-heurística-072a5fae145>. Acesso em: 10 maio 2025.
- FREITAS, Malu. **Algoritmo A* (A Estrela)**. [s.d.]*
Disponível em: <https://github.com/malufreitas/a-estrela>. Acesso em: 10 maio 2025.